

**MORPHOLOGICAL ANALYSIS-BASED
ARABIC SPELL CHECKING AND CORRECTION**

BY

TAMIM SALAH ALNETHARY

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

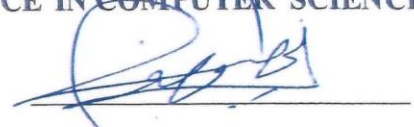
In

COMPUTER SCIENCE

January 2017

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **TAMIM SALAH ABDALLAH ALNETHARY** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE**.

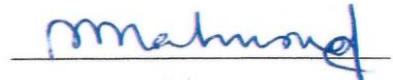


Dr. Wasfi G. Al-Khatib
(Advisor)



26/3/2017

Dr. Khalid A. Al-Jasser
Department Chairman



Prof. Sabri A. Mahmoud
(Co-Advisor)



Prof. Salam A. Zummo
Dean of Graduate Studies

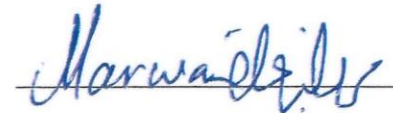


Dr. Lahouari Ghouti
(Member)

5/4/17
Date



Dr. Salahadin A. Mohammed
(Member)



Dr. Marwan Abu-Amara
(Member)

© Tamim Salah Abdallah Alnethary

2017

DEDICATED TO

I dedicate this dissertation with all of my love to my
parents, my lovely wife, my daughters, my brothers
and sisters.

ACKNOWLEDGMENTS

First and foremost I want express my deepest thanks to Allah who gave me strength, patience and ability to accomplish this thesis.

I wish to express my appreciation to Dr. Wasfi G. Al-Khatib and Prof. Sabri A. Mahmoud, who served as my major advisor and co-advisor, for their guidance and patience through the thesis, their support and encouragement can never be forgotten. Thanks are due to my thesis committee members Dr. Lahouari Ghouti , Dr. Salahadin A. Mohammed and Dr. Marwan Abu-Amara for their cooperation, comments and support. Thanks are also due to the Chairman of Information and Computer Science Department Dr. Khalid A. Al-Jasser for providing all the available facilities.

I would like to thank King Fahd University of Petroleum & Minerals (KFUPM) for supporting this research and providing the computing facilities. This work was supported by KACST NSTIP project 11-INF2159-04 “Arabic Spell Checking detection and correction”.

I also would like to thank Taiz University, which gave me the opportunity for completing my M.Sc. degree in KFUPM. I owe thanks to my colleagues and my friends for their motivation and pivoted support.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	VII
TABLE OF CONTENTS.....	VIII
LIST OF TABLES.....	XI
LIST OF FIGURES.....	IV
LIST OF ABBREVIATIONS.....	IV
ABSTRACT	V
ملخص الرسالة	VII
CHAPTER 1 INTRODUCTION.....	1
1.1 Spell Checking and Correction	1
1.2 Morphological Analysis	4
1.3 Problem Statement.....	6
1.4 Thesis Objectives and Outcomes.....	7
1.5 Thesis Outline	9
CHAPTER 2 LITERATURE REVIEW	10
2.1 Morphological Analysis	10
2.2 Spell Checking Detection and Correction.....	14
CHAPTER 3 MORPHOLOGICAL ANALYSIS & DISAMBIGUATION USING HMMS	23
3.1 Proposed Arabic Morphological Analysis Model	23
3.2 Preprocessing.....	26
3.3 Phase I: Morphological analysis	27

3.4	Phase 2: Features Disambiguation	32
3.4.1	Disambiguation Process Formulation	32
3.4.2	Estimation of Parameters	35
3.5	Experimental Setup and Results	38
CHAPTER 4 ERROR MODEL		41
4.1	Proposed Error Model	41
4.2	Error Correction Patterns Generator (ECPG)	42
4.3	Error-Correct Patterns Database (ECPD)	48
4.4	Correction Candidates Generator (CCG)	50
4.5	Experimental Setup and Results	51
CHAPTER 5 GENERAL SPELL CHECKING DETECTION AND CORRECTION		57
5.1	Baseline System	57
5.2	System Description	57
5.3	System Formulation	61
5.4	Estimation of Parameters	64
5.5	Experimental Setup and Results	67
5.5.1	Handling Non-word Errors	67
5.5.2	Handling Real-word Errors	71
CHAPTER 6 CONCLUSIONS AND FUTURE WORK		73
6.1	Conclusions	73
6.2	Future Directions	75
REFERENCES		77
APPENDICES		80

Appendix A. The Details Results of Morphological Analysis and Disambiguation	80
Appendix B. The Error Model Statistics Details	85
Appendix C. The GSpell Error Probability Distributions	91
Appendix D. The Overall System GUI Description.....	100
The Morphological analyzer and Disambiguater	101
The Error Model	104
The General Spell Checking Detection and Correction.....	107
VITAE.....	122

LIST OF TABLES

Table 1-1 pattern based stem extraction.....	5
Table 2-1 summary of morphological analysis and Disambiguation studies.....	14
Table 2-2 summary of Spelling detection and correction studies	21
Table 3-1 Example for morphological analysis and disambiguation phases.	24
Table 3-2 Mapping Dictionaries.....	29
Table 3-3 Sliding pattern “المربطون” over the input “مفتعل”.....	31
Table 3-4 Sample SWAM analyses for the word ’لأسرتهم’	32
Table 3-5 Summary of the morphological analysis, on NEMLAR corpus (Phase I).....	39
Table 3-6 Summary of morphological analysis disambiguation (Phase II)	40
Table 4-1 description of error encodings	44
Table 4-2 Examples of error patterns generated by ECPG	48
Table 4-3 examples of ECPD stored data	49
Table 4-4 Examples of error patterns candidates' generation.....	51
Table 4-5 QALB corpus errors summary	52
Table 4-6 Corpus error words statistics.....	53
Table 4-7 QALB error patterns summary	53
Table 4-8 Corpus error patterns statistics	53
Table 4-9 results of effectiveness of the error model based on QALB test data	55
Table 4-10 results of effectiveness of the minimum edit distance based on QALB test data.....	56
Table 5-1 Non-word error detection results	68
Table 5-2 Sample experiments with different window size and different model features.....	68
Table 5-3 Non-word error correction results (100% Detection)	69
Table 5-4 Non-word error correction results.....	70
Table 5-5 Real-word error probabilities distribution.....	72
Table 7-1 Result of morphological analysis using NEMLAR corpus.....	80
Table 7-2 Result of morphological analysis using NEMLAR corpus.....	82
Table 7-3 QALB Annotated Corpus Errors Statistics	86
Table 7-4 The statistics of QALB distinct error patterns	88
Table 7-5 Complete real-word error probabilities distribution.....	91
Table 7-6 Collected and used corpora.....	111
Table 7-7 Statistics of dictionaries	111
Table 7-8 Character N-grams	112
Table 7-9 LMs statistics of the corpus	113

LIST OF FIGURES

Figure 3.1 The Architecture of the proposed Arabic Morphological analyzer and Disambiguator model	25
Figure 3.2 SWAM matching engine.....	28
Figure 3.3 Graph resulting from phase I for each feature separately	35
Figure 4.1 Error-Correct patterns generator (ECPG) algorithm flowchart	43
Figure 4.2 Error-Correct patterns generator (ECPG) algorithm.....	45
Figure 4.3 Example of the ECPG for the words ('الرحمان' ، 'الرحمين')	46
Figure 4.4 EE and CC extraction process (STEP 4).....	47
Figure 4.5 Correction Candidates Generator (CCG).....	50
Figure 5.1 GSpell Error detection and correction.....	58
Figure 5.2 Examples of error detection and correction phases.....	60
Figure 5.3 Spelling correction hmm model	63
Figure 7.1 The System main GUI.....	100
Figure 7.2 The morphological analyzer main GUI.....	101
Figure 7.3 the MA output features and output directory	102
Figure 7.4 the MA output features.....	102
Figure 7.5 the HMM model setup	103
Figure 7.6 the morphological analysis result file	104
Figure 7.7 the error model	105
Figure 7.8 the error model train files and output directories	106
Figure 7.9 the error patterns result file snapshot	106
Figure 7.10 The GSpell main GUI	108
Figure 7.11 input/output setting of error text	109
Figure 7.12 HMM method setting.....	109

LIST OF ABBREVIATIONS

NLP	:	Natural Language Processing
ECPD	:	Error-Correct Patterns Database
ECPG	:	Error Correction Patterns Generator
CCG	:	Correction Candidates Generator
SWAM	:	Sliding Window Asynchronous Matching
POS	:	Part Of Speech
HMM	:	Hidden Markov Model

ABSTRACT

Full Name : Tamim Salah Abdallah Alnethary
Thesis Title : Morphological Analysis-based Arabic Spell Checking and Correction
Major Field : Computer Science
Date of Degree : **January 2017**

Spell checking is the process of locating spelling errors and possibly correcting them. The need for automatic spell checking detection and correction capabilities is vital in most state of the art text editing related applications. In this thesis, we address the problem of Arabic spell checking detection and correction for non-word and real-word errors. This is accomplished through the development of a morphological analyzer, an error patterns model and a hidden Markov Model (HMM) based language model.

A morphological analyzer that is based on the Sliding Window Asynchronous Matching (SWAM) algorithm was fully implemented and extended to provide morphological features for a running text. The morphological analyzer uses HMMs to disambiguate morphological features of the analyzed word based on context. The morphological analyzer functionalities are used to develop an error patterns model. The error model learns the error patterns of the Arabic language based on an already annotated error corpora. The error model generates and ranks candidate corrections for wide ranges of Arabic errors. It can also be used for analyzing error types for any error annotated corpora. These systems have been integrated into a general spell-checker prototype system that is capable of handling non-word and real-word-errors. In addition, previously developed non-word and real-word error detection and correction systems have also been integrated into the prototype system. The results of the morphological analyzer reported an accuracy of 97.13% for roots,

98.20% for stems and 95.80% for lemmas, based on NEMLAR corpus. In the case of the error model, the effectiveness of using the error model were evaluated using QALB error corpus. The results show that the model can help in the correction process for any spelling correction system with more than 84% coverage; this effectiveness can be improved by including more corpora in the learning process. The general spellchecker was evaluated using QALB and KFUPM corpora. Results of the general spellchecker are analyzed and future directions to improve the spellchecking detection and correction are provided.

ملخص الرسالة

الاسم الكامل: تميم صلاح عبدالله النظاري

عنوان الرسالة: التدقيق والتصحيح الإملائي للنص العربي باستخدام التحليل الصرفي

التخصص: علوم الحاسب الآلي

تاريخ الدرجة العلمية: يناير ٢٠١٧

التدقيق الإملائي هو عملية اكتشاف الأخطاء الإملائية مع إمكانية تصحيحها. وتعد الحاجة إلى التدقيق الإملائي التلقائي من الوظائف المهمة في كثير من التطبيقات وخاصة تلك المتعلقة بتحرير النصوص. في هذه الأطروحة حاولنا تناول مشكلة اكتشاف وتصحيح الأخطاء الإملائية وأخطاء الكلمات الحقيقية. ويتم ذلك باستخدام عدة أدوات طورت لهذا الغرض وهي: التحليل الصرفي ونموذج الأخطاء ونموذج اللغة المعتمد على نماذج ماركوف المخفية. قمنا بتطوير وتوسيع محلل صرفي (SWAM) للنص العربي. يقوم المحلل الصرفي باختيار التحليل المناسب وفقاً للسياق بالاعتماد على نماذج ماركوف المخفية (HMM). تم توظيف إمكانيات المحلل الصرفي لتطوير وبناء نموذج للأخطاء. يعتمد نموذج الأخطاء على عدة مكانز للغة العربية يتعلم من خلالها أنماط الأخطاء. يستطيع نموذج الأخطاء توليد وترتيب مقترحات لنطاق واسع من أخطاء اللغة، وكذلك يمكن استخدامه لتحليل ومعرفة أنواع الأخطاء اللغوية لأي مكانز. وتم كذلك اقتراح مدقق لمعالجة انواع متعددة من الأخطاء. المدقق الإملائي هو عبارة عن دمج للمحلل الصرفي ونموذج الأخطاء بالإضافة إلى خوارزمية للتصحيح تعتمد على نماذج ماركوف المخفية. وبالإضافة إلى ذلك، تم دمج نظامين لاكتشاف الأخطاء الإملائية وأخطاء السياق ليتم مقارنتهما مع المدقق الإملائي المقترح.

وقد أظهرت نتائج التحليل الصرفي نسبة دقة ٩٧,١٣% للجذور و ٩٨,٢٠% للجذوع و ٩٥,٨٠% بالنسبة للمصادر باستخدام المكنز (NEMLAR). بالنسبة لنموذج الأخطاء فقد أظهرت النتائج باستخدام المكنز QALB قدرة النموذج على مساعدة عملية تصحيح الأخطاء بنسبة ٨٤%، وتزداد هذه النسبة بزيادة حجم المكنز المستخدم لتعلم الأخطاء. تم كذلك تقييم أداء المدقق الإملائي باستخدام المكنز QALB و KFUPM وتم مناقشة النتائج وتوضيح آلية تطويرها في المستقبل.

CHAPTER 1

INTRODUCTION

1.1 Spell Checking and Correction

The problem of error detection and correction have been studied for decades. Many techniques were developed to solve this problem for English language. The area of Arabic Spell checking is not mature yet and no system achieved high error correction accuracy efficiently, including Microsoft word, which is the most widely used Arabic spell correction system [1, 2].

Spell checkers are classified into interactive and automatic systems [3]. Interactive systems include spell checkers that are able to detect and correct errors with the help of a user. Users are supposed to select the most accurate correction from a system-generated list of candidate corrections. Automatic systems on the other hand, include spell checkers that are able to detect and correct error words automatically without any user intervention.

The main components for any spell checking systems are: error detection, candidates' generation and error correction. The error detection component is responsible for detecting suspected errors in the input text. The candidates' generation component is responsible for generating a list of all probable corrections for each of the detected errors. The error correction component is responsible for selecting the most appropriate correction from the list of the generated candidates. The effectiveness of the whole system is highly affected

by the effectiveness of these components. The more accurate the detection process, the more likely that the errors will be corrected. On the other hand, the detected errors will be more probably corrected if the generated candidates are more likely to have the correct word. Any improvements in any component of the spell checking system can lead to significant cumulative improvements of the whole system.

The detection and correction process may differ based on the used evidences for achieving each component's task. The evidences are either to use the word form only, in the case of non-word errors, or to also consider the word context in the case of real-word errors. The simplest approach to use the word form is to use a dictionary or reference lists; any word that has no entry in the dictionary is flagged as non-word error. On the other hand, the simplest approach for using both the word form and context is to use n-gram statistics; if the inspected word does not usually appear in the current context surroundings words, it is flagged as probable error.

Correction candidates' generators calculate a set of similarity scores that helps in judging the similarity between two strings. The most commonly used approach for generating correction candidates is the edit distance [4]. The edit distance provides a measure of how two strings are similar. The similarity is defined by the minimum basic editing operations (insertion, deletion, substitution, and transposition) needed to transform an incorrect word into the correct word. The approach works by recursively calculating the edit distance between different substrings of an $M \times N$ matrix of the compared strings. This process is applied to all words in the used dictionary, although the dictionary may not cover all the words. It is a brute-force process that ends with a huge list of candidates with many possible repetitions and with no ordering of candidates having the same edit distance [5, 6]

Different types of errors can be detected and corrected by spell checkers viz. non-word and real-word errors [4, 7]. Non-word errors constitute words that do not exist in the dictionary of the spell checker such as “كتاب” for “كتابت”. Real-word errors on the other hand are dictionary words that have been mistakenly used in non-appropriate context, such as using the word “قيل” instead of “قتل” in the sentence “وقد قيل أكثر من عشرة أشخاص في هذا الهجوم”.

Arabic spelling errors arise from character changes that occur to the correct word. The source of these change operations is either typing errors or spelling errors. Typing errors (or typos) are errors that exist when the writer/typist erroneously presses different keyboard keys other than the intended one (الرحمان، الرحان). Spelling errors, on the other hand, are errors that exist as a result of the writer's ignorance of the correct spelling of the word. The main causes of this ignorance are phonetic similarity (ظباب، ضباب), semantic similarity (سيخسر، سبخس), lack of grammatical rules (سبع احجار، سبعة احجار), or dialect writing (حيخسر).

The errors may result in character insertions, deletions, substitutions, and transposition. Character insertion which occurs when an extra character is inserted into the intended word. For example, the letter ‘ت’ is inserted into the word ‘مكتوب’ where the intended correct word is ‘مكتب’. In Character deletion a character is deleted from the intended correct word. For example, the letter ‘ت’ is deleted from the word ‘استخدم’ resulting in a ‘اسخدم’. In character substitution a letter in the intended word is mistakenly substituted with another letter. For example, the letter ‘ش’ in the word ‘مدرشة’, erroneously substitutes the letter ‘س’ in the intended word ‘مدرسة’. The last change is character transposition, where two letters are swapped in the intended word. For example, in the word ‘اسخدم’, the letter ‘ت’ is erroneously swapped with the letter ‘خ’, in the intended word ‘استخدم’. If a single

character change is applied to the intended word, the error is called a simple error. Otherwise the error is called a complex error [4, 7].

1.2 Morphological Analysis

Morphological analysis is an important pre-processing step for Natural Language Processing (NLP) applications. The main goal of morphological analysis is to define words in terms of their morphosyntactic information such as word structure and part-of-speech (POS) [8]. This information is useful for many NLP applications such as parsing, POS tagging, spell checking and machine translation.

Word structure information includes the knowledge of the stem, the root, and the affixes. For English language, the terms stem and root are used interchangeably. The stem represents the part of the word that remains after removing the affixes. Affixes include prefixes and suffixes. Prefixes are attached to the beginning of the stem, while suffixes are attached to the end of the stem with some orthography rules. For example, the word *improperness* has the prefix "im" and the suffix "ness", and the stem "proper" [9, 10]. The word structure information for Semitic languages (like Arabic) includes the root, the stem, the lemma, the pattern and the affixes. The root of a word consists of the original letters from which the word is derived; while the lemma represents the dictionary form for a set of words. In [11], lemma is differentiated from the stem, although many researchers use them interchangeably. In our case, we also differentiate between a lemma and a stem, but only for the words that have some of their original letters deleted or transformed. For example,

the word كتابان share the same lemma and stem, while the word المهتدون has the stem مهتد and the lemma مهتدي.

A stem in Arabic can either be pattern-based (مشتق) or static (جامد). Pattern-based stems represent words that can be derived from a root following specific morphological pattern (rhyme). For example, the stem 'مرتبط' is derived from the root 'ر ب ط' following the morphological pattern 'مفعل' as shown in Table 1-1

Table 1-1 pattern based stem extraction						
Word	ا	ل	م	ر	ت	ب ط و ن
Pattern			م	ف	ت	ع ل
Affixes	ا	ل				و ن
Root				ر		ب ط
Stem			م	ر	ت	ب ط

The number of morphological patterns in Arabic is around 900 excluding their combinations with pronouns and external affixes. The number of Arabic roots is over 11000; apparently, 70% of which are Arabic roots that are tri-literal while the rest are quad-literal, [12].

A static stem represents the part of the word that is not derived using morphological patterns. It has one form in all its inflected words. It can be a nominal term like demonstrative, conditional, and circumstantial nouns such as 'إذا، من، فلان', etc..., or it can be verbal terms like 'عسى، ليس'.

Arabic is considered a morphologically-complex highly-inflectional language. Its root-pattern non-concatenative morphology makes computational processing expensive [13]. Some Arabic words may undergo certain character transformations during the derivation

process of their stem. This is quite common problem with words that belong to defective¹ and hamzated² roots. Almost 35% of Arabic words have roots that are defective and/or hamzated[8]. The root extraction process of such words has higher error rates than words belonging to the intact³ root category[8].

Moreover, there are many idiosyncrasies in Arabic. Some words do not follow the usual pattern formation process⁴, as in the case of Arabic broken plurals. This makes the lemma extraction process more difficult and requires the use of a dictionary in most cases. Furthermore, the underspecified Arabic orthography may create a high degree of ambiguity for processing Arabic text [13]. The ambiguity may also be a result of the implicit vocalization in Arabic. For example, when segmenting a word like 'عدتهم'; all the following segmentations ('عدّة + هم', 'عدّت + هم' and 'عدت + هم') are valid.

1.3 Problem Statement

The need for automatic spell checking detection and correction capabilities is vital in most state of the art text editing related applications. They are also important in correcting errors of Optical Character Recognition (OCR) output and on-line text recognition. The problem in Arabic language is the absence of a general system for detection and correction of Arabic spelling errors. Moreover, there is a lack of an automatic spelling corrector without the

¹ Defective roots are roots that contain one or more of the short vowel letters ('حروف العلة (ا، و، ي)') like 'قول', 'وعد', 'رمي', etc.

² Hamzated root are roots that have Hamza like 'أكل', 'سأل', 'قرأ', ...etc.

³ Intact roots are roots for the intact verbs (الأفعال الصحيحة) which are not defective verbs (الأفعال المعتلة)

⁴ For example, adding 'ات' to the noun 'معلم' give the noun female plurals 'معلمات', however with broken plurals this rule does not apply, e.g. the plural of the word 'كتاب' is 'كتب'.

need for human intervention, and without wasting much efforts and time when correcting in the traditional method.

1.4 Thesis Objectives and Outcomes

In this thesis, an Arabic morphological analyzer and disambiguater is developed to support the spell checking detection and correction process. The morphological analyzer provides a set of morphological features: the root, the stem, the lemma, the morphological pattern and the affixes. The morphological analyzer functionalities are utilized to build a novel error model prototype of Arabic errors. The error model prototype is able to generate and rank candidate corrections for wide range of Arabic errors. A general spell checker framework (GSpeller) is also proposed. The spell checker handle wide range of Arabic error types. This was achieved through an integration of the system components, a morphological analyzer, an error model, and a language model (HMM).

The outcomes of this thesis can be utilized in many scientific researches such as information retrieval (in the case of the morphological analyzer), spell checking (in the case of the error model), machine translation (in the case of the spell checker) etc. The main outcomes of this thesis are as follows:

- **Arabic morphological analyzer and disambiguater:**

A lexicon based broad coverage Arabic morphological analysis and disambiguation tool is developed. The tool can be used to support many NLP applications. In this work, the developed morphological analyzer and

disambiguater were used to support the Arabic spelling detection and correction process.

- **A novel error model for generating and ranking correction candidates:**

A data driven error model prototype that exploits morphological error patterns at the morphemes or the word levels is developed. The model learns the Arabic language error patterns from an already annotated error corpora. The model can be used to support the candidate's generation task for spelling correction system.

- **A proposed general spell checking framework:**

A spell checking prototype that handle wide range of Arabic error types is developed. The spell checker is developed through an integration between the system components, a morphological analyzer, an error model, and a HMM model.

- **Building a Baseline system**

Two previously implemented systems [4, 7] for spell checking detection and correction are integrated into a single system. The integrated system provides detection and correction of non-word and real-word errors.

- **Possible Publications:** The developed work is a result of research activities which have been integrated into different prototypes. The information reported in Chapter 3, Chapter 4 and Chapter 5 are suggested to be reported in papers for possible publication.

1.5 Thesis Outline

This thesis is organized as follows. The introduction with problem definition, research objectives and outcomes are discussed in the previous sections. The rest of the thesis is organized as follows. Chapter 2 presents a literature review of morphological analysis and spell checking detection and correction. In Chapter 3, the developed morphological analyzer and disambiguator is described. Chapter 4 presents the details of the error model prototype. Chapter 5 presents the details of the proposed general spell checking system. Finally, Chapter 6 concludes this thesis and summarizes the outcomes and future directions.

CHAPTER 2

LITERATURE REVIEW

The problems of error detection and correction have been studied for decades. Many techniques were developed to solve the problem for English language. The area of Arabic spell checking is not mature yet and no system achieved high results efficiently.

In this chapter, the currently existing work related to this thesis is surveyed. The chapter is divided into two parts. The work related to morphological analysis and disambiguation is presented in Section 2.1. The work related to spell checking detection and correction is presented in Section 2.2.

2.1 Morphological Analysis

Different Arabic morphological analyzers with different methodologies and tasks were developed to support Arabic NLP applications. Morphological analyzers can be classified into general purpose morphological analyzers, stemmers, and lemmatizers. General purpose Arabic morphological analyzers generate most possible analyses of the words out of their contexts. The most known Arabic general purpose morphological analyzers are: Buckwalter Morphological Analyzer (BAMA) [14] [15] and Alkhalil Morph System[16]. BAMA is a simple rule based morphological analyzer that depends on a set of lexicon lists. The lexicon lists include Arabic stems and stem-affixes compatibility tables. Morphological patterns are not included in BAMA. Alkhalil morphological system applies a set of morphosyntactic rules with the help of a set of linguistic resources to extract

more detailed morphological features. The generated morphological features include the word root, stem, affixes, possible patterns and vocalizations.

An open source morphological analyzer and POS tagger called Qutouf was developed by Altabba et al. [17]. This morphological analyzer modified and enriched Alkhalil morphsystem database. It was used for root extraction, pattern matching, morphological feature and POS assignment, and nouns list generation. A set of state machine automata was developed for fine-grain cliticization. A modification to Standard Arabic Language Morphological Analysis (SALMA) tagset[18] was designed and incorporated in the system to provide POS tagging using a rule based expert system.

Pasha et al. presented MADAMIRA, a morphological analysis and disambiguation tool that takes advantage of two previously existing tools MADA and AMIRA [19]. The tools use different language models with support vector machines (SVM). MADAMIRA can be used for stemming and POS tagging of large Arabic corpora.

Bounhas et al. presented an approach for disambiguating Arabic non-vocalized morphological features by combining Arabic classifiers and linguistic rules [20]. They perform unsupervised training for a set of unlabeled Arabic corpora. They provided two approaches for handling ambiguous features. A probabilistic classifier that directly handles the ambiguous features and a data-transformation classifier that allows converting ambiguous datasets into non-ambiguous ones. The linguistic rules are used to reduce the number of ambiguous features. The authors have suggested a method for handling out of vocabulary words with the help of the Levenshtein edit distance. The experiments show that the probabilistic approach performs better than the data-transformation approach. This

is because the associated morphological features with probabilistic approach have few (less than 6) class (POS) values.

Most existing Arabic morphological analyzers are just stemming algorithms (Stemmers)[8]. Stemmers extract stems/roots of the analyzed words according to their context, light stemmer extract the stem by direct stripping of affixes . Existing stemming algorithms fail to achieve accuracy rates of more than 75% [8]. They can be suitable for information retrieval applications where their accuracy do not affect their overall performance. However, accuracy is vital for other natural language processing applications such as spell checking and machine translation. The most known effective Arabic stemmers are Khoja [21] and Boudlal et al. [22]. Khoja stemmer is lexicon and rule-based stemmer that is designed as part of Khoja POS tagger. The stemmer removes the longest affixes and matches the remaining word with verbal and nominal patterns to extract the root. Khoja reported 96% stemming accuracy using newspaper text. Boudlal et al. used a data driven technique with statistical approach (HMM and Viterbi algorithm) for the root extraction process [22]. Accuracy of 98.31% is reported using NEMLAR Arabic writing corpus, [23] which is a manually annotated corpus

Ababneh et al. [12] used lists of affixes and patterns (with singular and plural patterns lists). Using samples from a list of terms, they compared their algorithm with the root-extraction stemmer (Khoja stemmer) and light stemmer (Larky stemmer). Their algorithm starts by matching the word with stored patterns to ensure that no affixes related to the word were removed. If there is no match with the pattern list, compatible affixes are truncated. Finally, the algorithm uses the list of singular and plural patterns to solve the problem of stemming plural forms of irregular nouns such as "مصانع" to "مصنع". According to the

authors, the stemmer was able to remove all affixes effectively without removing any part of the original word.

El-Defrawy et al. [24] described a context based Arabic stemmer (ABAS). They used a distributional semantics co-occurrence model for the task of selecting the most appropriate root. The distributional semantics utilize the Smoothed Pointwise Mutual Information (SPMI) to improve the disambiguation process. They reported 81% root extraction accuracy on a dataset of 10,302 words of the International Corpus of Arabic (ICA) [25].

Hadni et al.[26] used a mix of root-based, light and statistical stemming approaches. After normalizing the input, affixes were removed; then the results were matched against the stored patterns to extract the stem according to the word length. The resulting stem was then partitioned into bi-grams. The similarity between this and the bi-grams of all stored roots were computed. The most similar root was returned as the result. They used a list of 9000 roots, and other lists for affixes and patterns. Their experiment was conducted on the Corpus of Contemporary Arabic (CCA) [27]. They used 1450 Arabic words for testing. The reported average accuracies are: 74.41% for Khoja, 59.71% for light stemming, 48.17% for n-grams, and 82.33% for their method.

The much less studied Arabic morphological analyzers are the lemmatizers; Lemmatizers are responsible for extracting the lemma rather than the stem of the words. Among the works on Arabic lemmatizers is the one conducted by Aliwy [11]. Aliwy used a rule-based and statistical method with dictionary lookup for the lemma extraction. He reported 99.67% accuracy over his manually annotated corpus of 16K words.

The summary of the conducted studies on morphological analysis and disambiguation are shown in Table 2-1.

Table 2-1 summary of morphological analysis and Disambiguation studies

Study	Technique	Disambiguation	Data	Results
(Buckwalter 2002, 2004) [14] [15]	Lexicon Driven approach	No	NA	NA
(Boudlal 2010) [16]	Lexicon Driven approach	No	NA	NA
Qutouf [17] (Altabba et al. 2010)	Lexicon Driven approach	No	NA	NA
MADAMIRA (Pasha 2014) [19]	Support Victor machine	Yes	NA	NA
(Sawalha 2011) [8]	Lexicon Driven approach	No	A gold standard of Quran's (78,004 words)	NA
(Khoja 2002, 2004)[21]	lexicon and rule-based	No	newspaper text	96%
(Boudlal et al.2010) [22]	Data driven technique, HMM and Viterbi algorithm	Yes	NEMLAR	98.31%
Context based root-based stemmer (El-Defrawy 2015)[24]	distributional semantics co-occurrence model	Yes	10302 words	81%

2.2 Spell Checking Detection and Correction

The methods and approaches used for the spell checking task may differ based on the used evidences. The evidences are either to use the word form only, in the case of non-word errors, or to also consider the word context in the case of real-word errors. The simplest approach that use the word form is to use a dictionary or reference lists; any word that has no entry in the dictionary is flagged as non-word error. On the other hand, the simplest approach for using both the word form and context is to use n-gram statistics; if the inspected word does not usually appear in the current context surroundings words, it is flagged as probable error.

Correction candidates generator provides a list of suggested corrections for the suspected error words. The most commonly used approach for generating correction candidates is employing the edit distance. The edit distance provides a measure of how two strings are similar; the similarity is defined by the minimum basic editing operations (insertion, deletion, substitution, and transposition) needed to transform an incorrect word into the correct word. The approach works by recursively calculating the edit distance between different substrings of an $M \times N$ matrix of the compared strings. This process is applied to all words in the used dictionary, although the dictionary may not cover all the words. It is a brute-force process that ends with a huge list of candidates with many possible repetitions and with no ordering of candidates having the same edit distance [5, 6] .

Deorowicz and Marcin [28] apply a set of ranked string substitutions rules to generate the candidate corrections [28] for English language. The candidate's generation process was maintained with the help of acyclic deterministic finite automaton (ADFA) which allows quick rejection of nonsense corrections. Their method was compared with the correction list of Ispell, GNU Aspell and Microsoft Word built-in spellcheckers. Based on aspell⁵ and wikipedia⁶ datasets, they show that their method provide more accurate results for the first top-5 correction candidates.

Hamza, et al. proposed the use of a small size dictionary of stems to use in correcting spelling errors, instead of using a large dictionary[29]. This dictionary is similar to the one used by Buckwalter Aramorph morphological analyzer [14, 15]. For every input word that is not in the dictionary, a distance measure between the morphemes (prefix, stem and

⁵ A collection of hard-to-correct errors used for testing GNU Aspell.

⁶ A data set of typical spelling errors made by the editors of the Wikipedia Project.

suffix) of the analyzed word and the Aramorph lexicon tables is computed and the suggested correction is determined based on the minimum distances. Using a corpus of 2784 misspelled words, the authors reported that their method outperformed the classical Levenshtein approach in terms of the average time (0.10 and 0.19 for their method and Levenshtein approach respectively) and the correction rate (85% and 50% for their method and Levenshtein approach respectively).

Nejja and Abdellah [5] used the concept of surface patterns and roots with the Levenshtein minimum edit distance to generate corrections of the error word. He provides a correction process using three approaches that are mainly based on selecting the surface pattern of minimum edit distance with the correct word. The approaches were compared to the Levenshtein edit distance. By using a dictionary of 10000 automatically generated misspelled words. The system reported 94.42, 95.42% and 93.34% correction rates and 0.1418ms, 0.1659ms, and 0.1519ms execution times which is better than Levenshtein edit distance correction rate and execution time (77.38% and 0.1953ms).

Zaghouani et al. [30] used regular expressions with a set of hand written linguistic rules for the correction process. The approach works by replacing a predefined set of errors with their suggested corrections based on a set of rules. The rules are built based on manual inspection of the nature of native and non-native errors. Using a test data provided from QALB shared task, the authors reported F1 measure correction accuracy of 66.9% for native speaker's data and 31.72% for non-native speaker's data.

Hicham et al.[31] introduced an approach for correcting errors that result from insertion, deletion and substitution operations. When ranking candidates, they used the frequency of

change operations with the Levenshtein distance. Using a list of 190 error words, they reported a correction accuracy of 62.63% and 10% for their method and Levenshtein distance respectively.

Mays et al. [32] proposed an unsupervised technique for real word error detection and correction based on word tri-grams. They used a word vocabulary of 20,000 words. For each word in their word vocabulary, they generated a confusion set. For training, they used word tri-gram probabilities provided by IBM speech recognition project. For testing, they selected randomly 100 valid sentences from the AP newswire and transcript of the Canadian Parliament. These 100 sentences are used to generate a group of 8628 sentences with a single real-word error in each sentence. They reported that their technique achieved a precision of 76% for detection and 73% for correction.

Wilcox-O'Hearn et al [33] evaluated the advantages and limitations of Mays [32] technique. Based on the evaluation, they proposed an improved version of the technique. In their version, they increased the vocabulary of the tri-gram model and shortened sentence window size. Besides that, multiple errors rather than single error in each sentence are considered. The obtained results show that the improved version performed poorer than their original technique.

Islam et al. [34] suggested the use of Google Web IT 3-grams dataset for multiple real word errors detection and correction. In their technique, a group of 500 articles is used to evaluate the performance of their technique. These articles are collected from Wall Street Journal corpus. They reported a recall of 89% for detection and 76% for correction. They stated that Google 3-grams are proved useful for real-word error detection and correction.

Three different techniques were proposed for error detection and correction by Majed [7]. Two of these techniques were supervised techniques, context was considered based on word co-occurrence method and the n-gram language model. The third one was unsupervised technique. The unsupervised technique was based on N-grams languages models. This technique was developed to exploit N-grams language models to compute their probability for the error detection and correction.

A multi agent system for semantic errors detection was introduced by Zerbi et al. [35]. The system combined co-occurrence, co-occurrence collocation, vocabulary vector, and latent semantic analysis. Four contextual methods were used to represent words within sentences in terms of their contexts. Then, a voting procedure was used to select the most probable error. A corpus extracted from the Egyptian newspaper Al-Ahram was used for training and testing. They reported an accuracy of 86.46%, 82.95%, 81.96%, 72.30% and 62.12% for Voting, co-occ-collocation, Latent Semantic Analysis (LSA), co-occurrence and vocabulary vector, respectively.

In Tomeh et al. [36], an approach that pipelined character and word-level translation model with re-ranking and punctuation insertion model were used for the correction of QALB corpus as part of the first Arabic shared task for error detection and correction. They reported 58.6% error correction F1 score with 76.9% and 47.3% recall and precision, respectively.

A set of correction rules that maximized the overall F-score was calculated from the training data was presented Nawar and Ragheb [37]. The system was developed for the

EMNLP2014 shared task for Arabic automatic error correction. They reported 65% error correction F-score on the provided QALB test data.

Shalan et al. [38] used Buckwalter morphological analysis results as an error detection component in their system. A rule-based and distance based mechanisms are then used to help in the correction process. A set of 190 misspelled words were used for testing the system. They reported 80% and 90% recall and precision respectively.

Hassan et al. [1] proposed a system that targets the detection and correction of several error types of QALB shared task corpus, including edit, add before (punctuations), merge and split errors. The system detect erroneous words by applying Buckwalter morphological analyzer. For each detected “Edit” and “add before” errors, classifiers with contextual features are used to correct them. A random insertion and omitting of a space were maintained to correct merge and split errors. They reported 58% F1 error correction score with 59% and 58% recall and precision, respectively.

Shalan et al. [39] created a large-coverage word list for Arabic of 13 million words, with 9 million having fully inflected valid surface words using AraComLex⁷. A character-based tri-gram language model was created from valid and invalid forms. A context-free finite-state automaton for measuring the edit distance between input words and the suggested corrections was created. Candidates were ranked based on scores generated from a noisy channel model trained on a corpus of one-billion words and knowledge-based rules of common errors.

⁷ An open-source finite-state large-scale morphological transducer.

Hassan et al. [40] proposed a language independent method for detecting and correcting spelling errors. Error detection is based on finite state automata while candidates' generation was handled using a Levenshtein-transducer that is compatible with the finite state machine. Ranking candidates was handled using the n-gram language model. He reported an accuracy of 89% based on Arabic and English text.

Rozovskaya et al. [41] presented an error annotation framework, as part of QALB (Qatar Arabic Language Bank) joint project, that aims to build a large manually corrected Arabic corpus text for building automatic correction tools. This corpus is useful for spell checking applications. The corpus now contains 1.2 million annotated words. A portion of the corpus was released to the participants of the Arabic error correction shared task at the EMNLP 2014 Arabic NLP workshop[41].

A context-based system was suggested to automatically correct misspelled words by Alkanhal et al. [42]. The misspelled words are firstly ranked using the Levenshtein edit distance considering space insertion and space deletion errors. The most correct candidate for each misspelled word is then selected according to the maximum marginal probability via A* lattice search and N-gram probability estimation. They reported an improved performance reaching F-scores of 97.9% and 92.3% for detection and correction, respectively, based on their manually annotated corpora.

Attia et al. [6] used Levenshtein edit distance with a knowledge-based rules to re-order the number of candidates. Based on his experiment, the amount of noise present in the training data has the potential effect in the improvement of the results. Using a test set of 2,027

spellings error, the presented system outperforms Ayaspell, MS Word, and Google Docs for the ranking of candidates in first position.

The summary of the conducted studies on spelling detection and correction are shown in Table 2-2.

Table 2-2 summary of Spelling detection and correction studies

Work	Technique	Candidate Generation	Test Data	Target errors	Results
Hamza, et al. 2014 [29]	Morphological analysis with Levenshtein edit distance	minimum edit distance for word morphemes	2784 misspelled words	Non-word	85%
Nejja and Abdellah 2014 [5]	NAN	surface patterns and roots with the Levenshtein edit distance	10,000 misspelled words and 2000 random misspelled words	Non-word	93%
Zaghouni et al. 2015 [30]	regular expressions with a set of hand written linguistic rules	a combination of pre-existing tools, hand written contextual rules and lexicons	QALB	NAN	F1 of 66.9% for native speakers' data and an F1 of 31.72% for the non-native speakers' data
Hicham et al. 2012[31]	NAN	the frequency of change operations with the Levenshtein distance	190 errors of a typing Arabic documents for a set of users.	Non-word	62,63%
(Majid 2013) [7]	Word co-occ. and n-gram	Levenshtein edit distance	27K from Al-Arabiya Website	Real-word	Detection F-score 20.2 Correction F-score 17.3
(Zribi , Ahmed 2013, 2007) [35]	Voting, co-occ. Colloc., LSA, co-occ. and voca. vector	NAN	Random errors in Al-Ahram Egyptian newspaper corpus	Real-word	Detection 86.46%, 82.95%, 81.96%, 72.30% and 62.12%
Tomeh et al. 2014 [36]	character and word-level translation model	a weighted finite-state transducer	QALB	NAN	58.6% F1 , 76.9% recall and 47.3% precision
Nawar and Ragheb 2014 [37]	probabilistic correction rules that maximized the overall F-score	The learnt rules	QALB	NAN	Correction F-score 65.7%
Shaalán et al. 2010 [38]	Buckwalter morphological analysis and semantic features	a rule based edit distance and a heuristic rule-based transformation	A set of 190 misspelled words	Non-word and Real-word	80% and 90% recall and precision respectively
Hassan et al. 2014 [1]	Buckwalter and classifiers with contextual features for correction	Levenshtein edit distance	QALB	NAN	58% F1 ,59% recall and 58% precision

Shaanan et al. 2012 [39]	Dictionary look up and finite-state automaton and the noisy channel model	finite-state automata of Levenshtein distance	400,000 words with 6,279 misspelled words by MS Spell Checker	Non-word	precision of 98.2% at a recall of 100%
Hassan et al. 2008 [40]	finite state automata and n-gram language model	Levenshtein-transducer	A list of 11,000 words	Non-word	89%
Alkanhal et al. 2012 [42]	Dictionary lookup A* lattice search and N-gram probability estimation	Damerau–Levenshtein distance	Different annotated corpora	Non-word	97.9% detection 92.3% correction F-score
Attia et al. 2012 [6]	knowledge-based re-ranking rules and NGrams language model	Levenshtein edit distance	2,027 spellings errors	Non-word	82.86 %

CHAPTER 3

ARABIC MORPHOLOGICAL ANALYSIS AND DISAMBIGUATION USING HMMS

Arabic morphological analysis and disambiguation received interest as an active area of research. Most existing morphological analyzers are based on root or stem extraction. The reported results of such techniques are not satisfactory. In this chapter, a lexicon based morphological analyzer and disambiguater is presented. The system can be described as a root-based stemmer, lemmatizer, and morphological pattern extractor. The system extracts these features for any word by considering its context using Hidden Markov Model (HMM).

3.1 Proposed Arabic Morphological Analysis Model

The main goal of this model is to generate the most probable morphological features for each word in an input text. The features include the word root, stem, lemma, morphological pattern and affixes. The overall architecture is shown in Figure 3.1. The input text is preprocessed using normalization and tokenization which is described in section 3.2. The morphological analysis and disambiguation process is accomplished in two separate phases. In Phase 1, all possible morphological analyses are generated for each word in the input text using a data driven matching approach as described in Section 3.3. Each analysis consists of a tuple of features as shown in Table 3-1. In the second phase, a Markovian-

based Viterbi algorithm selects the best tuple of features for each word in the input text based on its context as described in Section 3.4.

Table 3-1 Example for morphological analysis and disambiguation phases.

Word	Phase	Features					
		SRoot	ORoot	Stem	Lemma	AffixSPattern	AffixOPattern
وجدت	I	وجد	وجد	وجدت	وجدت	فعل \$ ت \$	فعل \$ ت \$
		جد	جدد	جد	جدد	و \$ فل \$ ت	و \$ فعل \$ ت
		جد	جود	جد	جود	و \$ فل \$ ت	و \$ فعل \$ ت
		وجد	وجد	وجد	وجد	\$ فعل \$ ت	\$ فعل \$ ت
	II	وجد	وجد	وجدت	وجدت	و \$ فل \$ ت	و \$ فعل \$ ت
أسرتهم	I	أسر	أسر	أسرت	أسرة	\$ فعلت \$ هم	\$ فعلة \$ هم
		أسر	أسر	أسرت	أسرت	\$ فعلت \$ هم	\$ فعلت \$ هم
		سر	سرر	أسر	أسرر	\$ أفع \$ تهم	\$ أفع \$ تهم
		سر	سرر	أسرت	أسررت	\$ أفعت \$ هم	\$ أفعت \$ هم
	II	أسر	أسر	أسرت	أسرة	\$ فعلت \$ هم	\$ فعلة \$ هم
الدليل	I	دلل	دلل	دلّيل	دلّيل	ال \$ فعيل \$	ال \$ فعيل \$
	II	دلل	دلل	دلّيل	دلّيل	ال \$ فعيل \$	ال \$ فعيل \$

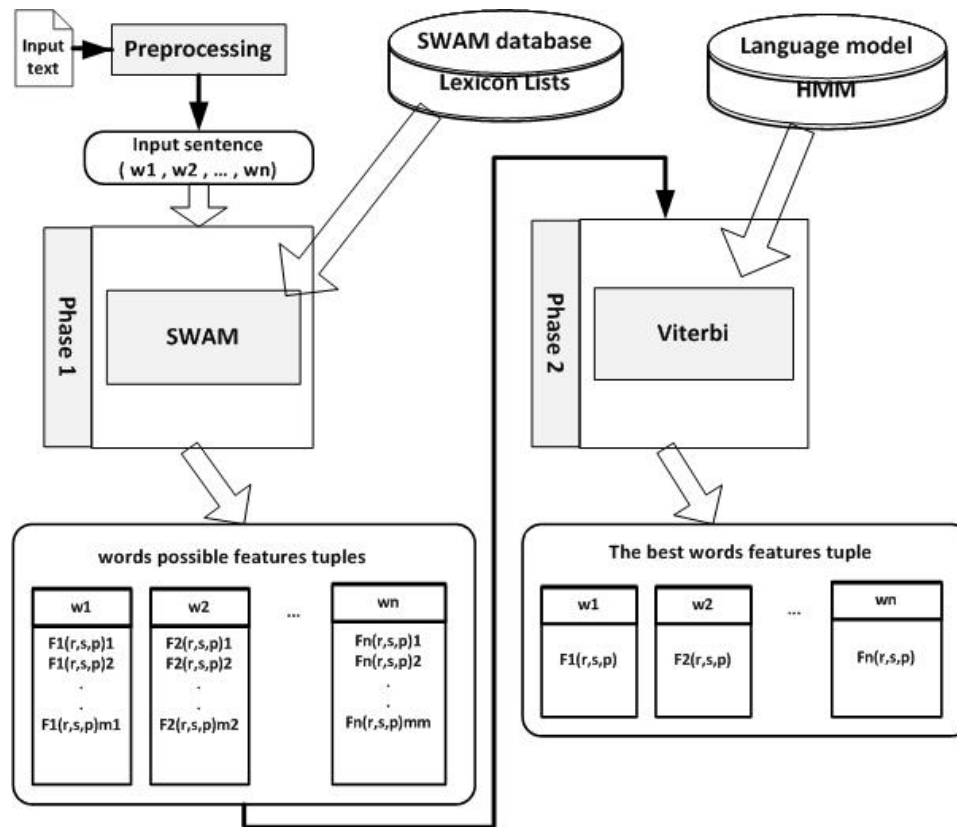


Figure 3.1 The Architecture of the proposed Arabic Morphological analyzer and Disambiguator model

3.2 Preprocessing

Most NLP systems require preprocessing of the input text before applying any word-level NLP tasks. Two preprocessing steps are applied in this system; tokenization and normalization. The tokenization process refers to the assignment of each word in the input text to a specific class or type, called token. Hence, the input text is split into a set of consecutive sentences; each of which contains a sequence of tokens. In this work, the following tokens are considered: a word, a number, a named entity, a multiword expression, and punctuation marks.

Normalization aims to remove noise that may exist in the input text, as a results of different writing styles. The following normalization steps are applied:

- Removal of the elongation (Tatweel) letter '-' in words. For example, 'المــــأذن' is changed to 'المأذن'.
- Removal of all diacritics. For example, 'المأذن' is changed to 'المأذن'.
- Replacement of Alif Madda 'آ' with two Alif 'أأ'. For example, 'المأذن' is changed to 'المأذن'.

It is worth mentioning that some authors use an additional normalization step where the letters 'ا', 'إ' and 'آ' are changed to 'ا' ; also the letter 'ة' is normalized into 'ه' and the letter 'ى' is normalized into 'ي' [13]. However, we chose not to include this step as the proper mapping of the letters are already included in the system's lexicon list.

3.3 Phase I: Morphological analysis

The objective of Phase I is to generate all the probable analyses for each word in the input text. Each analysis includes the following features: the root, the stem, the lemma, the morphological pattern and the affixes. The analyses are generated using a lexicon based matching algorithm, SWAM. The algorithm examines a set of lexicon lists and returns all the possible analyses (features tuples) that match the input word.

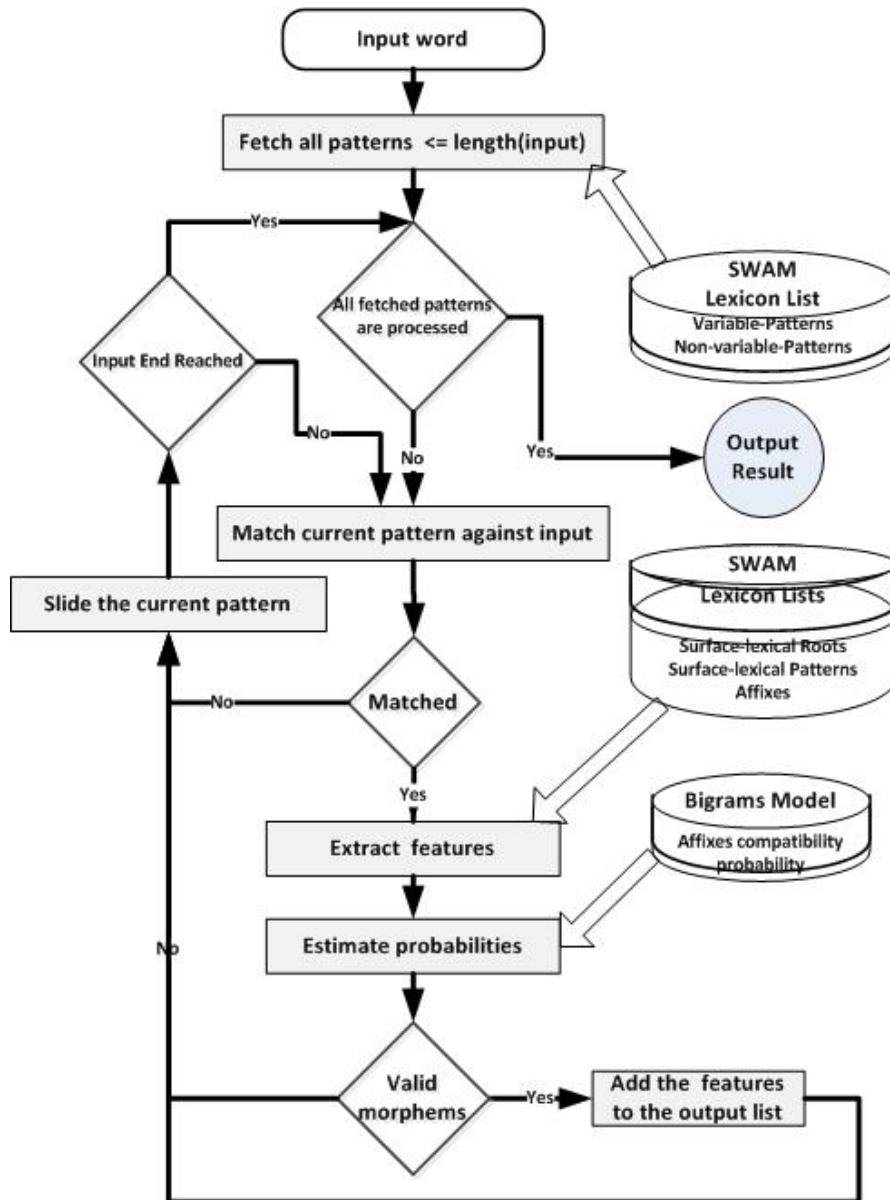


Figure 3.2 SWAM matching engine

SWAM uniformly processes fixed⁸ and variable⁹ words. The SWAM algorithm is based on the differentiation between lexical forms and surface forms. Lexical forms refer to the traditional roots, variable patterns and fixed words that are usually stored in classical lexicons. The term “surface form” is used for the lexical forms that have possibly undergone changes, during the word formation process, resulting in one or more alteration to their letters. This differentiation provides implicit information to handle the idiosyncrasies of Arabic. SWAM keeps both lexical and surface forms in its lexicons using dictionaries. Surface forms are used as mapping keys into their corresponding lexical forms. SWAM includes a list of possible prefixes, e.g. (“ال، ف، س”) and a list of possible suffixes e.g. (“ان، ون، ين”). Table 3-2 shows some examples of the lexical and surface forms of sample roots, patterns of words and fixed words.

Table 3-2 Mapping Dictionaries

Word	Mapping	
	Surface	Lexical
Root	خذ	أخذ
	قل	قول
	قال	قول
	هدي	هدي
	هد	هدي
Pattern	أففع	أففع
	فواع	فواع
	انفع	انفع
	تفعل	تفعل
	تفل	تفعل
Fixed	ذل	ذلك
	الذي	الذي
	من	الذي
	لتي	الذي
	فلان	فلان
	على	على
	لولا	لو

⁸ A fixed word is a word that does not have a pattern-based stem, referred to as 'جامد'.

⁹ A variable word is a word that has a pattern-based stem, referred to as 'مشتق'.

The original existing SWAM lexicon lists were not enough to do experiments, many items in the lists were missing. This forces us to collect all the required list in order for SWAM to work perfectly. In this regards, we generated the required lists from Alkhalil morphological system. The generated lists were large and required an extensive revision. However, we handled a manual revision for the generated lexicon lists, the lists still have some extra elements which somehow affect the results.

Given a particular input word, the algorithm extracts members of the above lists that produce an acceptable decomposition of the input word. This is achieved by sliding the stored surface patterns (with the same length or less than that of the input word) against the input word and computing the resulting decompositions at each position. A decomposition is acceptable if all the resulting components are valid and have compatible morphemes. At this point, the affixes compatibility is handled by determining the existence of at least one occurrence of the full pattern with compatible affixes in the lists.

An example of the sliding window process is shown in Table 3-3 for the word 'المرتبطون' with the pattern 'مفتعل'. It is obvious that Row 3 contains the only matching pattern. The prefix-suffix pair (denoted by suffix\$prefix), which is ال\$ون, are compatible with the pattern.

Table 3-3 Sliding pattern “مفتعل” over the input “المرتبطون”

Sliding Window										Action	Matching	#
Position	9	8	7	6	5	4	3	2	1	slide	Mismatch	1
Input Word	ن	و	ط	ب	ت	ر	م	ل	ا			
Pattern					ل	ع	ت	ف	م			
Root	*	*	*	*	*	*	*	*	*			
Position	9	8	7	6	5	4	3	2	1	slide	Mismatch	2
Input Word	ن	و	ط	ب	ت	ر	م	ل	ا			
Pattern				ل	ع	ت	ف	م				
Root	*	*	*	*	*	*	*	*	*			
Position	9	8	7	6	5	4	3	2	1	extract	Match	3
Input Word	ن	و	ط	ب	ت	ر	م	ل	ا			
Pattern			ل	ع	ت	ف	م					
Root	*	*	ط	ب	*	ر	*	*	*			
Position	9	8	7	6	5	4	3	2	1	slide	Mismatch	4
Input Word	ن	و	ط	ب	ت	ر	م	ل	ا			
Pattern		ل	ع	ت	ف	م						
Root	*	*	*	*	*	*	*	*	*			
Position	9	8	7	6	5	4	3	2	1	End	Mismatch	5
Input Word	ن	و	ط	ب	ت	ر	م	ل	ا			
Pattern	ل	ع	ت	ف	م							
Root	*	*	*	*	*	*	*	*	*			

Different accepted decompositions may result when applying SWAM to an input word. They may be generated from one or more surface patterns. Examples of multiple decompositions are shown in Table 3-4 for the word 'الأسرتههم'. Note the huge difference in meaning for each analysis. In this case, the best decomposition can be determined through

the disambiguation process that comprises Phase 2 of the system, as we describe in the next section.

Table 3-4 Sample SWAM analyses for the word 'أسرتهم'

root	Lemma	Suffix-Pattern- Prefix (Surface Form)	Suffix-Pattern- Prefix (Lexical Form)	English meaning of the Word
أسر	أسرة	ل - فعلت - هم	ل - فعلة - هم	family
أسر	أسرت	ل - فعلت - هم	ل - فعلت - هم	seize
سرر	أسيرة	ل - أفعت - هم	(ل - أفعه - هم)	beds

3.4 Phase 2: Features Disambiguation

The features disambiguation process attempts to determine the best tuple of features that correspond to a word from the generated list of feature tuples produced by the matching algorithm in the first phase. The most appropriate feature tuple for each word in the input text depends highly on its context. The best features tuple for each word is selected using a Markovian based Viterbi algorithm as explained in the next sections.

3.4.1 Disambiguation Process Formulation

Let $W = \{w_1, w_2, \dots, w_m\}$ represent the set of Arabic words, and let the sets $R = \{r_1, r_2, \dots, r_k\}$, $S = \{s_1, s_2, \dots, s_k\}$, $L = \{l_1, l_2, \dots, l_l\}$, $AP = \{a_1, a_2, \dots, a_h\}$ and $FP = \{p_1, p_2, \dots, p_h\}$ represent the roots, stems, lemmas, affix patterns¹⁰ and full patterns¹¹ of Arabic words, respectively. Given an Arabic sentence $S = (w_1, w_2, \dots, w_n)$, $w_i \in$

¹⁰ By affix pattern, we mean the original word pattern along with affixes separated by \$, for example the affix pattern for the word 'المهتدون' is '\$المفتعل\$ون'

¹¹ By full pattern, we mean the original word pattern along with affixes, for example the full pattern for the word 'المهتدون' is 'المفتعلون'

$W, 1 \leq i \leq n$, and let the set of morphological analyses generated by Phase I for each word $M(w_i) = \{f_{i_1}, f_{i_2}, \dots, f_{i_{m_i}}\}$ where word w_i has m_i possible analyses. Each analysis f_{i_j} is a tuple of 5 features $(r_j, s_j, l_j, a_j, p_j)$ where $r_j \in R, s_j \in S, l_j \in L, a_j \in AP$ and $p_j \in FP$. The goal is to find the most likely feature tuples $f^* = (f^*_1, f^*_2, \dots, f^*_n)$ of the sentence S , where $f^*_i \in M(w_i)$. This can be formulated as follows:

$$f^* = (f^*_1, f^*_2, \dots, f^*_n) = \argMax(p((f_1, f_2, \dots, f_n)|(w_1, w_2, \dots, w_n))) \quad (2)$$

According to Bayes rule:

$$p((f_1, f_2, \dots, f_n)|(w_1, w_2, \dots, w_n)) = \frac{p((w_1, w_2, \dots, w_n)|(f_1, f_2, \dots, f_n)) \times p(f_1, f_2, \dots, f_n)}{p(w_1, w_2, \dots, w_n)} \quad (3)$$

Substituting (3) in (2) we get:

$$f^* = \argMax \left(\frac{p((w_1, w_2, \dots, w_n)|(f_1, f_2, \dots, f_n)) \times p(f_1, f_2, \dots, f_n)}{p(w_1, w_2, \dots, w_n)} \right) \quad (4)$$

The prior probability of the word sequence $p(w_1, w_2, \dots, w_n)$ in (4) is a positive constant and is independent of the features. Therefore, it has no influence on the ranking of the different sequences and can be ignored, this reformulates our goal as:

$$f^* = \argMax(p((w_1, w_2, \dots, w_n)|(f_1, f_2, \dots, f_n)) \times p(f_1, f_2, \dots, f_n)) \quad (5)$$

Assuming that the probability of any word features depends only on the features that precede it (Markov assumption)¹², and the probability of the word depends only on its features (Markov output independence assumption)¹³, we can reformulate our goal as:

$$f^* = (f^*_1, f^*_2, \dots, f^*_n) = \argMax(\prod_{i=1}^n p(w_i|f_i) \times \prod_{i=1}^n p(f_i|f_{i-1})) \quad (6)$$

The resulting statistical model in (6) is called the Hidden Markov Model (HMM), where the input words represent the model observations and feature tuples represent the hidden states. The best features sequence in (6) is computed using the Viterbi algorithm. For computing the best features sequence, a supervised learning module was built for estimating the HMM parameters as described in the next section.

¹² $p(t_k|t_1 \dots t_{k-1}) = p(t_k|t_{k-1})$, this follows that $p(t_1 \dots t_k) = \prod_{i=1}^k p(t_i|t_{i-1})$

¹³ $p(w_k|t_k, w_{k-1}, t_{k-1}, \dots w_1, t_1) = p(w_k|t_k)$, this follow that $p(w_1 \dots w_n|t_1 \dots t_n) = \prod_{i=1}^n p(w_i|t_i)$

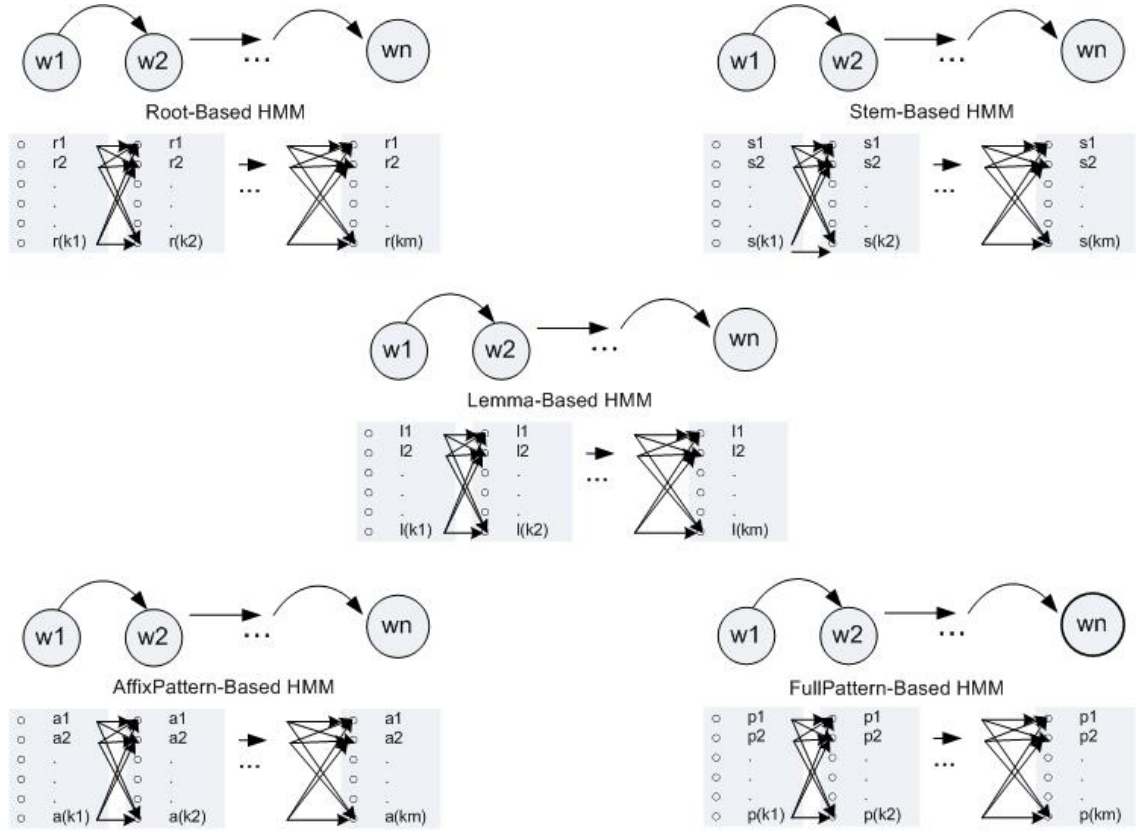


Figure 3.3 Graph resulting from phase I for each feature separately

3.4.2 Estimation of Parameters

A supervised learning module was built for estimating the HMM parameters, viz., the transition probabilities $Prob\{f_i|f_{i-1}\}$ and the emission probabilities $Prob\{w_i|f_i\}$. The probabilities were estimated from an already-tagged corpus, using the maximum likelihood estimation method, in two phases. In the first phase, the probabilities (emission and transitions probabilities) for each feature (root, stem, lemma, affix pattern or full pattern) are estimated separately as shown in Figure 3.3; zero-probabilities were smoothed using add- λ smoothing with backoff method (Christopher and Schuetze 1999). λ was set higher for words or features that rarely occur, since the training data may not contain rarely-

occurring words and/or features in the language. In the second phase, the total smoothed probabilities of the whole feature tuple are then estimated as the product of the probabilities in the first phase, as shown in Equations 11 and 12. This is done under the assumption that the probability given a specific feature is independent from the probability given any other feature. This provides a customized estimation of the parameters for any feature or combinations of features.

The overall smoothed transition probabilities $Prob\langle f_i | f_{i-1} \rangle$ is defined as:

$$Prob(f_i | f_{i-1}) = \prod_{j=1}^m \frac{c(f_{(i-1)j}, f_{ij}) + \lambda_j \cdot Prob_{ff_backof}(f_{ij} | f_{(i-1)j})}{c(f_{(i-1)j}) + \lambda_j} \quad (11)$$

Where

- f_{ij} represent the j^{th} morphological feature of the features tuple f_i
- $c(f_{(i-1)j}, f_{ij})$ is the number of times feature $f_{(i-1)j}$ appears in the training corpus followed by feature f_{ij}
- $c(f_{(i-1)j})$ is the number of times the feature $f_{(i-1)j}$ appears in the training corpus
- λ_j is the number of j^{th} feature types such that $c(f_{(i-1)j}, f_{ij}) = 1$

The overall emission probabilities $Prob\langle f_i | f_{i-1} \rangle$ is defined as:

$$Prob(w_i | f_i) = \prod_{j=1}^m \frac{c(f_{ij}, w_i) + \lambda_j \cdot Prob_{wf_backoff}(w_i | f_{ij})}{c(f_{ij}) + \lambda_j} \quad (12)$$

where

- $c(f_{ij}, w_i)$ is the number of times the word w_i appears in the training corpus with feature f_{ij}
- λ_j is the number of the word types such that $c(f_{ij}, w_i) = 1$

In Equations 11 and 12, m represents the number of features considered in the features tuple, where the maximum in our case is equal to 5, viz., root, stem, lemma, affix pattern and full pattern. λ_j was assigned very small number, e.g. $(1e - 10)$, in the case when the number of singletons is zero ($\lambda_j = 0$). Since the sizes of the sets of roots, stems, lemmas, etc. vary greatly, the best value of λ_j in each case is determined during the system validation process as described in the next section.

The backoff estimates for each feature are defined as follows:

$$Prob_{ff_backoff}(f_{ij}|f_{(i-1)j}) = \frac{c(f_{ij})+1}{n_j} \quad (13)$$

$$Prob_{wf_backoff}(w_i|f_{ij}) = \frac{c(w_i) + 1}{n_j + V} \quad (14)$$

The above backoff estimate uses add-one smoothing; where n_j denotes the number of j^{th} feature that were observed in the training data, and V denote the number of words that were observed in the training data. Some words that does not occur in the train data may happen to exist in the test data as novel words; these words are treated as if they had been replaced in the input by an out of vocabulary word, denoted by OOV, which is added to the set V . On the other hand, some features that does not occur in the train data may be generated by the morphological analyzer in the first phase as novel features; these features are treated as if they had been replaced by a single special feature that we call out of state, denoted by

OOS, which is added to the value n_j , since the features are suggested by the morphological analyzer, not from the training data features.

3.5 Experimental Setup and Results

The morphological analysis and disambiguation process was evaluated using a manually annotated corpus, NEMLAR¹⁴. Since stems and lemmas were not provided by NEMLAR, Stems were extracted by stripping off affixes from the word, and lemmas were extracted by matching each word against its annotated pattern. The corpus was then partitioned into training and testing data. The training data contains 346298 words, 76.4% of the corpus, while the testing data contains 106676 words, and 23.6% of the corpus. To study whether the value of lambda has any effect on the performance of the system, the 10-fold cross validation algorithm was run on different very small values of lambda for each feature based on the training data. It was observed that varying lambda did not have any effect on the performance. Therefore, we set the value of lambda to 10^{-10} .

In order to study how features affect generating a correct morphological analysis, different sets of features were tested, as shown in the tables of the Appendix. The percentages under Phase I column, in those tables, indicate the percentage of the existence of the correct morphological analysis among all generated analyses for each given word.

Each column under Phase II corresponds to selected sets of one or more features used in the HMM to disambiguate one or more morphological analyses. Each set of morphological analyses occupies one row in the table, with the last row indicating the average

¹⁴ Manually annotated words from journalistic Arabic texts of different categories. Each word in the corpus was annotated with its prefix, root, pattern and suffix. The corpus has almost 500K words

performance of each HMM feature set in successfully selecting the right analyses. The first table in the appendix was generated without consideration of affixes compatibility, unlike the second table, where affixes compatibility was taken into account. The sets of features considered in the design of HMMs and/or the sought morphological analysis include one or more combinations of root, stem, lemma, Full Pattern¹⁵ and Affixes Pattern¹⁶.

Table 3-5 Summary of the morphological analysis, on NEMLAR corpus (Phase I)

Analyses	Without Compatibility		With Compatibility	
	% Accuracy	Average # of Analyses	% Accuracy	Average # of Analyses
Lemma (L)	98.98	20	98.78	12
Root (R)	99.93		99.66	
Stem (S)	99.95		99.65	
Affix Surface Pattern (ASP)	98.99		98.57	
Affix Original Pattern (AOP)	98.58		98.77	
Root (R), Stem (S), Lemma (L), Affix Surface Pattern (ASP), Affix Original Pattern (AOP)	97.70		97.69	

Table 3-5 shows a summary of the results of Phase I with and without compatibility checking. In addition, the average number of generated analyses for each word in the testing data has been included in the table. It is clear from the table that the correct analysis exists among the generated set of analyses for each test word. Although the reported

¹⁵ By FullPattern, we mean the word pattern along with affixes , for example the affix pattern for the word 'المفتعلون' is 'المهتدون'

¹⁶ By AffixPattern, we mean the word pattern along with affixes separated by \$, for example the full pattern for the word 'المهتدون' is 'ال\$مفتعل\$ون'

accuracy of the analyses without compatibility checking is slightly higher than those with compatibility checking, it is obvious that the average number of generated analyses using compatibility checking is lower than that without it, since analyses with incompatible affixes are not considered. This affects the disambiguation when using compatibility checking, as the number of possibilities to choose from is less.

Table 3-6 shows part of the data in the tables of the Appendix corresponding to the chosen set of analyses. The column marked with "×" shows the results without compatibility checking, and the column marked with "✓" shows the results with compatibility checking. For example, the third row in the table shows the accuracy of determining the right stem when using all the features, only the root, only the stem or only the lemma, respectively, in the disambiguation process. It is obvious that the best results were achieved when using all features. In this case, the improvement achieved when using compatibility checking was not that high. However, it was significantly higher in most other cases.

Table 3-6 Summary of morphological analysis disambiguation (Phase II)

Analyses	AOP, SP,S,L ,R		Root		Stem		Lemma	
	×	✓	×	✓	×	✓	×	✓
Lemma (L)	95.80	95.80	55.24	68.49	33.54	42.87	94.30	95.02
Root (R)	97.13	96.93	96.27	96.62	53.99	58.18	82.96	84.25
Stem (S)	98.20	98.02	75.90	87.67	97.16	97.67	85.19	89.97
Affix Surface Pattern (ASP)	95.13	95.20	46.99	63.13	33.88	44.70	70.57	76.55
Affix Original Pattern (AOP)	96.03	96.07	55.63	72.28	44.01	53.16	65.76	71.42
Root (R)&Stem (S)&Lemma (L)& Affix Surface Pattern (ASP)&Affix Original Pattern (AOP)	94.11	94.16	44.41	59.33	28.36	37.36	63.18	68.63

CHAPTER 4

ERROR MODEL

Correction candidates' generators calculate a set of similarity scores that helps in judging the similarity between two strings. The most commonly used approach for generating correction candidates is employing the edit distance [4]. The edit distance provides a measure of how two strings are similar; the similarity is defined by the minimum basic editing operations (insertion, deletion, substitution, and transposition) needed to transform an incorrect word into correct word. The approach works by recursively calculating the edit distance between different substrings of an $M \times N$ matrix of the compared strings. This process is applied to all words in the used dictionary, although the dictionary may not cover all the words. It is a brute-force process that may end with a large list of candidates with many possible repetitions and with no ordering of candidates having the same edit distance [5, 6].

In this chapter, a data driven approach that exploits morphological error patterns at the morphemes or the word levels is proposed. The model is able to generate and rank candidates' correction for wide range of Arabic errors.

4.1 Proposed Error Model

Arabic is considered one of the morphologically rich languages, since many of its words are derived from a finite set of morphological patterns. This fact can be utilized in generating smarter candidate words for correcting spelling errors that may follow certain patterns. It is highly desirable that any such model possesses the following properties:

1. The correction candidates must be generated in a way where the number of candidates is relatively small, yet containing the correct word.
2. The model must be general, and not specific to certain type of spelling errors.
3. The model must provide a ranking for the correction candidates, such that the correct word is preferably at the top of the list.

In order to achieve these goals, a data driven approach that exploits morphological error patterns at the morphemes or the word levels is proposed. The main components of the model are the error-correct patterns generator (ECPG), the error-correct patterns database (ECPD), and the correction candidates' generator (CCG). The ECPG is a module that generates morphological error patterns and their correction information that are used in the correction process. The information generated by the ECPG is used to build the ECPD which is used by the CCG to generate the correction candidates. The ECPG is presented in Section 4.2, the ECPD is described in Section 4.3 and the CCG is detailed in Section 4.4.

4.2 Error Correction Patterns Generator (ECPG)

The ECPG module generates all the error patterns with their correction information from an already annotated error corpus with the help of a morphological patterns generator. In our work, we have utilized the SWAM morphological analyzer that is described in chapter 3. The overall algorithm of ECPG is outlined in Figure 4.1 Error-Correct patterns generator (ECPG) algorithm flowchart shown in Figure 4.1. For any error word, the corresponding correction is morphologically analyzed, using SWAM, to get its morphological pattern and affixes. For example, given the real word error 'الرحمين' in the context of 'أسباب الرحمين من'. The correct word 'الحرمان' is analyzed using SWAM, identifying it as 'ال\$حرم\$ان' with

morphological pattern 'ال\$فعل\$ان'. This information and the actual error word are used to generate the error pattern 'ال\$عفل\$ين'.

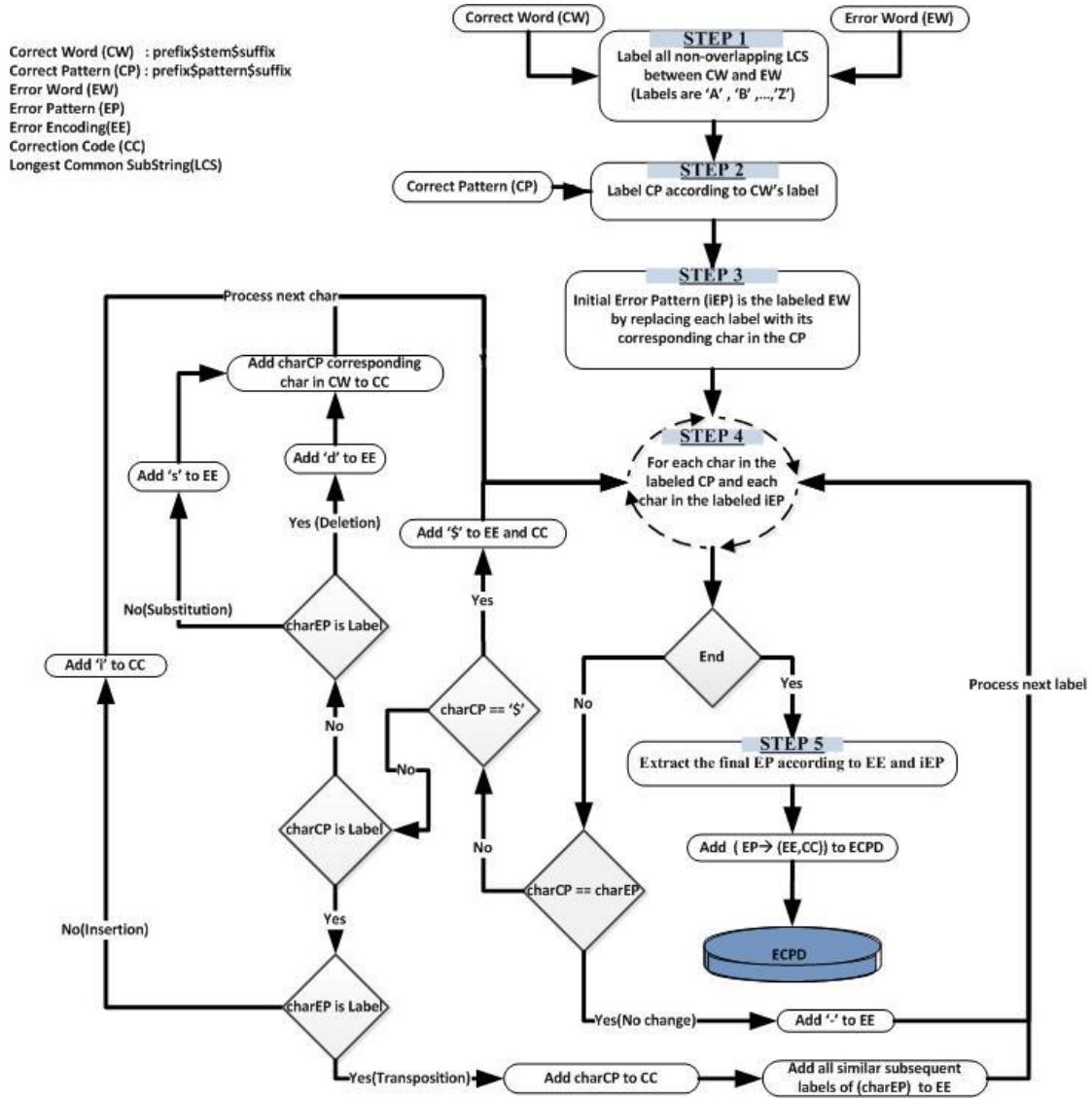


Figure 4.1 Error-Correct patterns generator (ECPG) algorithm flowchart

An error encoding is also generated, which is a string that specifies the positions of the changes and the change types in the error pattern. In our example, we have two changes:

transposition of the first two characters of the surface pattern and a substitution of the first character in the suffix. Hence, its error encoding (EE) is '-s\$-BC\$--', according to the error types listed in Table 4-1. Finally, the corrections that need to be applied are generated, which specify the actual changes. These corrections are denoted by the correction codes (CC). In our example, the correction code is '\$CB\$', which specifies the transposed and substituted characters. The ECPG algorithm is described in Figure 4.2, and the steps of the algorithm applied to our previous example are illustrated in Figure 4.3 and Figure 4.4, with Figure 4.1 showing the details of Step 4 of the algorithm.

Table 4-1 description of error encodings

EE (Error Encoding)	CC(Correction Code)	Error operation
-	"	No change
i	"	Insertion
d	'Arabic char'	Deletion
s	'Arabic char'	Substitution
Latin characters	'Latin char'	Transposition

The generated error patterns, error encodings and correction codes are stored in the ECPD. This database will be, later, used by the CCG to generate the correction candidates.

Algorithm ErrorCorrectPatternGenerator:

Input: Correct Word (**CW**), Error Word (**EW**) and Correct Pattern (**CP**)

Output: Error Pattern (**EP**), Error Encoding (**EE**) and Correction Code (**CC**)

- Step 1.** Label all non-overlapping Longest Common Substrings (LCS) between **CW** and **EW**
(Choose Labels from English Alphabets A, B, C,..., Z)
- Step 2.** Label **CP** according to **CW**
- Step 3.** The initial **EP** (**iEP**) is the labeled **EW** by replacing each label with its corresponding char in **CP**
- Step 4.** Loop over the whole letters of CP and iEP
 - a. If there is No Change :
 - i. Add '-' to **EE**
 - b. If there is letter insertion:
 - i. Add 'i' to **EE**
 - c. If there is letter deletion:
 - i. Add 'd' to **EE**
 - ii. Add the deleted letter to **CC**
 - d. If there is letter substitution:
 - i. Add 's' to **EE**
 - ii. Add the substituted letter to **CC**
 - e. If there is letter transposition:
 - i. Add the transposed label to **EE**
 - ii. Add the correct label to **CC**
- Step 5.** Extract the final **EP** according to **EE** and **iEP**

Figure 4.2 Error-Correct patterns generator (ECPG) algorithm

Step 1									
CW	ن	ا	\$	م	ر	ح	\$	ا	ل
EW			ن	ي	م	ح		ر	ا
CW Label	E	ا	\$	D	C	B	\$	A	A
EW Label		E	ي	D	B	C		A	A

Step 2									
CW Label	E	ا	\$	D	C	B	\$	A	A
CP	ن	ا	\$	ل	ع	ف	\$	ا	ل
CP Label	E	ا	\$	D	C	B	\$	A	A

Step 3									
CP	ن	ا	\$	ل	ع	ف	\$	ا	ل
CP Label	E	ا	\$	D	C	B	\$	A	A
EW Label		E	ي	D	B	C		A	A
iEP		ن	ي	ل	ف	ع		ا	ل
iEP Label		E	ي	D	B	C		A	A

Step 4									
CP Label	E	ا	\$	D	C	B	\$	A	A
iEP Label		E	ي	D	B	C		A	A
EE	-	s	\$	-	B	C	\$	-	-
CC			ا	\$	C	B	\$		

Step 5									
EE	-	s	\$	-	B	C	\$	-	-
iEP			ن	ي	ل	ف	ع		ا
EP			ن	ي	\$	ل	ف	ع	ا

Figure 4.3 Example of the ECPG for the words ('الرحمان' ، 'الحرمين')

<div>CP iEP</div>	A	A	\$	B	C	D	\$	'	E
A	'-':''								
A		'-':''							
C			'\$':'\$'	'C':'B'					
B					'B':'C'				
D						'-':''			
¢							'\$':'\$'	's':'l'	
E									'-':''

Figure 4.4 EE and CC extraction process (STEP 4)

Table 4-2 Examples of error patterns generated by ECPG

Input			Patterns		
Number	Error Word	Correct Word	Suffix	Stem	Prefix
			CSP ←ESP CC:EE	CP←EP CC:EE	CPP←EPP CC:EE
1	المرتبط	المرتبطون	” ←ون dd:ون	مفتعل ← مفتعل ”:-----	ال ← ال ”:--
2	كثير	كبير	” ←” ”.”	فثيل ← فثيل ”-s--ب	” ←” ”.”
3	الاسلام	الإسلام	” ←” ”.”	افعال ← إفعال ”-s---إ	ال ← ال ”:--
4	فالطالف	الطالب	” ←” ”.”	ففاعل ← فاعل ”-i---	فال ← ال ”-i--
5	هذب	ذهب	” ←” ”.”	عفل ← فعل CB:BC-	” ←” ”.”
6	مخابر	مخبر	” ←” ”.”	مفاعل ← مفعل ”-i---	” ←” ”.”
7	حيشرب	سيشرب	” ←” ”.”	يفعل ← يفعل ”:-----	ح ← س س:S
8	كتابب	كتاب	” ←ب ” :i	فعال ← فعال ”:-----	” ←” ”.”
9	ممثلي	ممثلين	ي ← ين -d:ن	مفعل ← مفعل ”:-----	” ←” ”.”

Table 4-2 shows samples of different error patterns that affect prefixes (4 and 7), stems (2-6) and suffixes (1, 8 and 9).

4.3 Error-Correct Patterns Database (ECPD)

ECPD is a database that holds the error patterns with their correction information. The database is created by, first, including the patterns generated by the ECPG. Then, for each correct stem (i.e. not containing any error), the algorithm examines all combinations of prefixes and suffixes, where at least one of them (the prefix or the suffix) has an error. If the combination satisfies the condition that the stem is compatible with the correct affixes,

such combination will be added to the ECPD. For incorrect stems, the same procedure applies with the difference that both affixes can be correct. Again, the compatibility check is only considered against the correct stem and correct affixes. For example, Table 4-3 contains error patterns that are generated from the first three rows of Table 4-2.

Table 4-3 examples of ECPD stored data

Error pattern (ESP\$EP\$EPP)	Correction Information		
	Correct Pattern (CSP\$CP\$CPP)	EE	CC
\$فالمفتعل\$	\$المفتعل\$	i--\$----\$	\$ \$
\$المفتعل\$	\$المفتعل\$ون	--\$----\$dd	\$ \$ون
\$فالمفتعل\$	\$المفتعل\$ون	i--\$----\$dd	\$ \$ون
\$مفتعل\$ب	\$مفتعل\$	\$----\$i	\$ \$
\$المفتعل\$ب	\$المفتعل\$	--\$----\$i	\$ \$
\$فالمفتعل\$ب	\$المفتعل\$	i--\$----\$i	\$ \$
\$مفتعل\$ي	\$مفتعل\$ين	\$----\$-d	\$ \$ن
\$المفتعل\$ي	\$المفتعل\$ين	--\$----\$-d	\$ \$ن
\$فالمفتعل\$ي	\$المفتعل\$ين	i--\$----\$-d	\$ \$ن
\$فثيل\$	\$فعيل\$	\$-s--\$	\$ب\$
\$المفثيل\$	\$المفعيل\$	--\$-s--\$	\$ب\$
\$فالمفثيل\$	\$المفعيل\$	i--\$-s--\$	\$ب\$
\$فثيل\$	\$فعيل\$ون	\$-s--\$dd	\$ون\$ب\$
\$المفثيل\$	\$المفعيل\$ون	--\$-s--\$dd	\$ون\$ب\$
\$فالمفثيل\$	\$المفعيل\$ون	i--\$-s--\$dd	\$ون\$ب\$
\$فثيل\$ي	\$فعيل\$ين	\$-s--\$-d	\$ن\$ب\$
\$المفثيل\$ي	\$المفعيل\$ين	--\$-s--\$-d	\$ن\$ب\$
\$فالمفثيل\$ي	\$المفعيل\$ين	i--\$-s--\$-d	\$ن\$ب\$
\$فثيل\$ب	\$فعيل\$	\$-s--\$i	\$ب\$
\$المفثيل\$ب	\$المفعيل\$	--\$-s--\$i	\$ب\$
\$فالمفثيل\$ب	\$المفعيل\$	i--\$-s--\$i	\$ب\$
\$افعال\$	\$إفعال\$	\$-s---\$	\$إ\$
\$المافعال\$	\$المإفعال\$	--\$-s---\$	\$إ\$
\$فالمافعال\$	\$المإفعال\$	i--\$-s---\$	\$إ\$
\$افعال\$ي	\$إفعال\$ين	\$-s---\$-d	\$ن\$إ\$
\$المافعال\$ي	\$المإفعال\$ين	--\$-s---\$-d	\$ن\$إ\$
\$فالمافعال\$ي	\$المإفعال\$ين	i--\$-s---\$-d	\$ن\$إ\$
\$افعال\$ب	\$إفعال\$	\$-s---\$i	\$إ\$
\$المافعال\$ب	\$المإفعال\$	--\$-s---\$i	\$إ\$
\$فالمافعال\$ب	\$المإفعال\$	i--\$-s---\$i	\$إ\$

The error and correct patterns information are stored using dictionaries according to the error pattern length. Searching for an error pattern is, therefore, started by computing the length of the error word and considering the dictionary corresponding to that length.

4.4 Correction Candidates Generator (CCG)

The correction candidates generator is a data driven module that uses error patterns to generate correction candidates for a given word. It simply considers the correction information stored at the EPCD that correspond to the error pattern. Figure 4.5 show the algorithm flowchart of CPG.

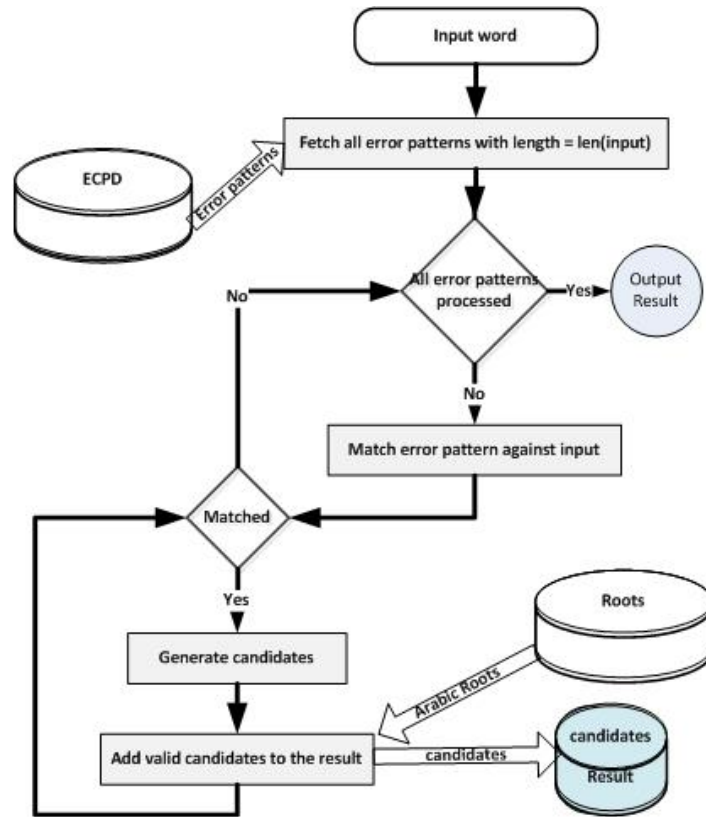


Figure 4.5 Correction Candidates Generator (CCG)

Given a particular error word, e.g. 'فالمستمع', the algorithm matches all the morphological error patterns, similar in length to the input, against the input word and generate all the correction candidates according to the matched patterns correction information. The matching occurs if all the morphological pattern letters other than ('ف', 'ع', 'ل') match their corresponding letters in the error word. Non-valid candidates are automatically rejected as possible candidates; a correction is valid if it has a valid root. According to the database in Table 4-3, only the pattern 'فالمفتعل' provide valid candidates.

Table 4-4 shows that only two correction patterns (المفتعل ، المفتعلون) are possible based on the given database in Table 4-3

Table 4-4 Examples of error patterns candidates' generation

Error Word								Correction Information					
								CSP	CP	CPP	Root	Correction	Valid
EWord	ع	م	ت	س	م	ال	ف		مفتعل	ال	سمع	ال\$مستمع\$	True
								ون	مفتعل	ال	سمع	ال\$مستمع\$ون	True
Pattern	ل	ع	ت	ف	م	ال	ف						

4.5 Experimental Setup and Results

To study how effective the suggested error model, an analysis of some well-known Arabic annotated corpora is conducted. The analysis is concentrated on the types and number of edits needed to get the correct word from the error word. The corpus chosen for this task is QALB. QALB is a manually annotated errors' corpus consisting of user comments collected from Al-Jazeera News webpage. It is mainly written in Modern Standard Arabic with almost one Million words with 243,075 errors. QALB has its specific format. However, we converted its format into KACST format. The corpus files are then fed to the

morphological analyzer (SWAM) to get the morphological pattern of the corpus words. The corpus has different types of errors, some of errors are annotated with more than three words as correction. In this work we include only sentences that has, among its errors, less than four connected words; the reason behind this exclusion is that such errors are almost unique to occur and they introduce overhead over the morphological analyzer disambiguation process.

Table 4-5 shows summary of the QALB corpus errors based on the error model analysis result file. Need to mention that the deletion error type ratio (40.18) in Table 4-5 include the percentage of 'punctuation deletion'¹⁷ which is 33.45, see Appendix Table 0-3 for more details.

Table 4-5 QALB corpus errors summary

Error Type	QALB	
	Count	Ratio
Insertion	19270	7.93
Deletion	97668	40.18
Substitution	106702	43.90
Transposition	530	0.22
Mixed¹⁸	18905	7.78
No Change	23	0.01
Totals	243075	100

Most of the errors in QALB are repeated errors, the total number of distinct corpus errors is 89536 as shown in

Table 4-6. The total distinct errors are composed of uniquely occurring errors (Non-Repeated error words, 68253) and the distinct number of repeated errors (21283).

¹⁷ 'Punctuation deletion' refer to the error that result when the writer miss to include a punctuation after specific word. In QALB corpus, this type of error is called 'add before'

¹⁸ Mixed error refer to complex error (more than one error operations)

Table 4-6 Corpus error words statistics

Error Type	QALB	
	Count	Ratio
Non-Repeated Corpus Errors	68253	%76.23
Repeated Corpus Errors (Distinct)	21283	%23.77
Total Distinct Corpus Errors	89536	100%

The summary statistics of the QALB corpus error patterns based on the error model analysis result file are shown in Table 4-7.

Table 4-7 QALB error patterns summary

Error Type	QALB	
	Count	Ratio
Insertion	6834	63.34
Deletion	17786	39.33
Substitution	9965	47.32
Transposition	414	86.97
Mixed	11130	92.83
No Change	19	95.00
Totals	46129	51.52

The details of the error words and error patterns for each change operation is provided in Appendix Table 0-3 and Table 0-4

Table 4-8 Corpus error patterns statistics

Error Type	QALB	
	Count	Ratio
Non-Repeated Error Patterns	32826	36.66
Repeated Error Patterns (Distinct)	13303	14.86
Total Distinct Error Patterns	46129	51.52

The total number of distinct generated error patterns is 46129 as shown in Table 4-8. The generated total distinct error patterns are composed of uniquely generated patterns (Non-Repeated error patterns, 68253) and the distinct number of repeated error patterns (21283). With the fact that the total number of the analyzed corpus errors is 243075; the statistics show that almost 71% ($243075 - 68253 = 174822$) of these errors are just a repetition of

only 21283 distinct errors with an average of almost 8 occurrences per error word. The remaining 29% of the corpus errors are singly occurred errors (non-repeated errors). This means that any upcoming error word should have their correction candidate in the list (if we use a simple non -error patterns model) with the probability of at least 71%.

Although, the number of uniquely occurred errors in the corpus is 68253 which is almost triple the number of distinct repeated errors; the total number of the whole generated error patterns by the error model is 46129 (51.52% of the whole distinct corpus errors).

We also conducted an analysis of the actual error model candidates' generation effectiveness on QALB test corpus. The experiment concentrated on the ability of the error model, based on the learnt error patterns, to generate the correct candidates among all the suggested candidates. The ranking of candidates was handled based on the error pattern repetition frequency. Different training corpora were used to train the model based on different error patterns frequencies. For example, Table 4-9 shows that, when training the model with QALB training corpus and including only the error patterns repeated 3 times or more, 83% of the correct corrections, of QALB test corpus, does exist in the generated candidates. The average number of candidates is 20 and the average candidates' generation time is 0.11. Moreover, almost 0.17% of the corrections exist among the top correction.

Table 4-9 results of effectiveness of the error model based on QALB test data

Training Corpus		Avg. # of generated candidates	Avg. # of error correction to exist as 1-Top candidate	Avg. # of error correction to exist as 5-Top candidate	Avg. # of error correction to exist as 10-Top candidate	Avg. # of error correction to exist in all candidates	Avg. candidates generation time (second)
QALB	all	50.626	0.175	0.719	0.789	0.847	0.421
	3	20.192	0.174	0.718	0.787	0.834	0.108
	5	17.760	0.174	0.718	0.786	0.832	0.137
	10	14.650	0.174	0.717	0.770	0.812	0.098
	20	11.753	0.174	0.709	0.760	0.798	0.069
	50	8.802	0.172	0.699	0.748	0.772	0.080
	100	6.892	0.171	0.452	0.410	0.512	0.044
KFUPM	all	4.288	0.203	0.320	0.326	0.326	0.053
	3	1.555	0.202	0.299	0.304	0.304	0.022
	5	1.149	0.200	0.298	0.298	0.298	0.019
	10	0.817	0.186	0.271	0.271	0.271	0.021
	20	0.745	0.178	0.252	0.252	0.252	0.013
	50	0.618	0.170	0.241	0.249	0.241	0.011
	100	0.521	0.168	0.236	0.236	0.236	0.011
QALB & KFUPM	all	51.780	0.207	0.723	0.795	0.851	0.363
	3	20.725	0.207	0.723	0.792	0.839	0.129
	5	18.137	0.207	0.722	0.791	0.837	0.177
	10	14.990	0.206	0.722	0.775	0.818	0.120
	20	12.054	0.206	0.713	0.767	0.801	0.082
	50	8.988	0.2041	0.706	0.757	0.780	0.045
	100	7.140	0.203	0.460	0.508	0.520	0.064

In the case of Levenshtein minimum edit distance, the candidates were generated using an edit distance of 1 and 2 as in Table 4-10. The experiment concentrated on the ability of the Levenshtein minimum edit distance to generate the correct candidates among all the suggested candidates. The candidates with the required edit distance are generated from a dictionary¹⁹ with 125975 words. We did not include the candidates generation within the edit distance of 3 since the generation time was very long (it takes more than 33 minutes/error word) and the memory requirement were very huge (4800 candidates/error word).

¹⁹ The dictionary was generated from different corpora by including word with more than 5 occurrences

Table 4-10 results of effectiveness of the minimum edit distance based on QALB test data

Training Corpus		Avg. # of generated candidates	Avg. # of error correction to exist in all candidates	Avg. candidates generation time
Edit Distance	1	19.1279	0.817781	3.423594
	2	440.641	0.856714	9.772839

CHAPTER 5

GENERAL SPELL CHECKING DETECTION AND CORRECTION

5.1 Baseline System

The baseline system is a combination of two previously implemented systems[4, 7] for spell checking detection and correction. It handles non-word and real-word spelling errors using different techniques. In the case of non-word, the system use the combination of Buckwalter Arabic Morphological Analyzer, dictionary look-up or Character N-grams. In the case of real-word errors the system use NGrams language model or context co-occurrence with confusion sets. More details of the subsystems and their combinations are described in the Appendix D.

5.2 System Description

The main goal of the described prototype is to effectively detect and correct wide range of Arabic errors. This is achieved through an effective interaction between the system components (viz. a morphological analyzer, an error model, and a language model). The morphological analyzer generates morphological features of the running text. The used morphological analyzer is SWAM. SWAM morphological features include the root, the stem, the pattern and the affixes. The error model generates the probable corrections for suspected error words, the used error model is ECPD (Error Correct Patterns Database).

The language model provides markovian based statistical description of the language based on the same morphological features generated by the morphological analyzer.

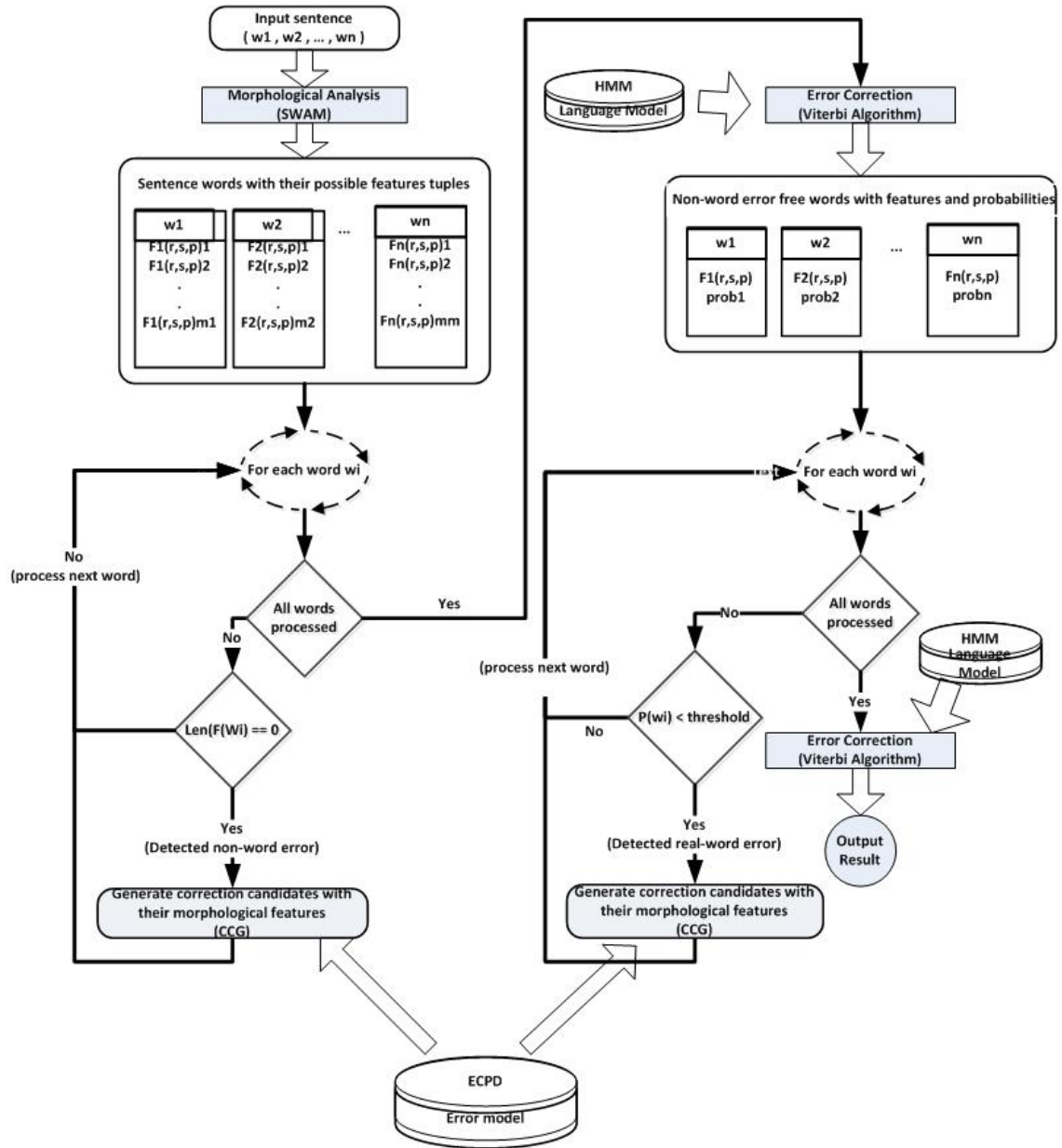


Figure 5.1 GSpell Error detection and correction

The system achieve the task of error detection and correction in two interleaved phases, each phase handle part of the problem as shown in Figure 5.1

In the first phase, the error detection and correction task start by morphologically analyzing all words in the input. Any word with no morphological analysis is flagged as suspected non-word error. The correction candidates with their morphological features are generated for each suspected non word using the CCG (Correct Candidates Generator). Under the assumption that the probability of any input word to happen in some context is highly related to the probability of its morphological features in the same context. The probabilities are computed for each word based on its morphological features. The word with the highest probability is selected by the correction algorithm as the best correction for the non-word error.

In the second phase, any word "with probability less" than a threshold is flagged as suspected real-word error. The correction candidates with their possible morphological features are generated for each suspected real-word error. The word with the highest morphological features probability is selected by the correction algorithm as the best correction of the real-word error. Figure 5.2 provides an example of how the detection and correction process work in the two phases. The detailed description of the algorithm is given in the next section.

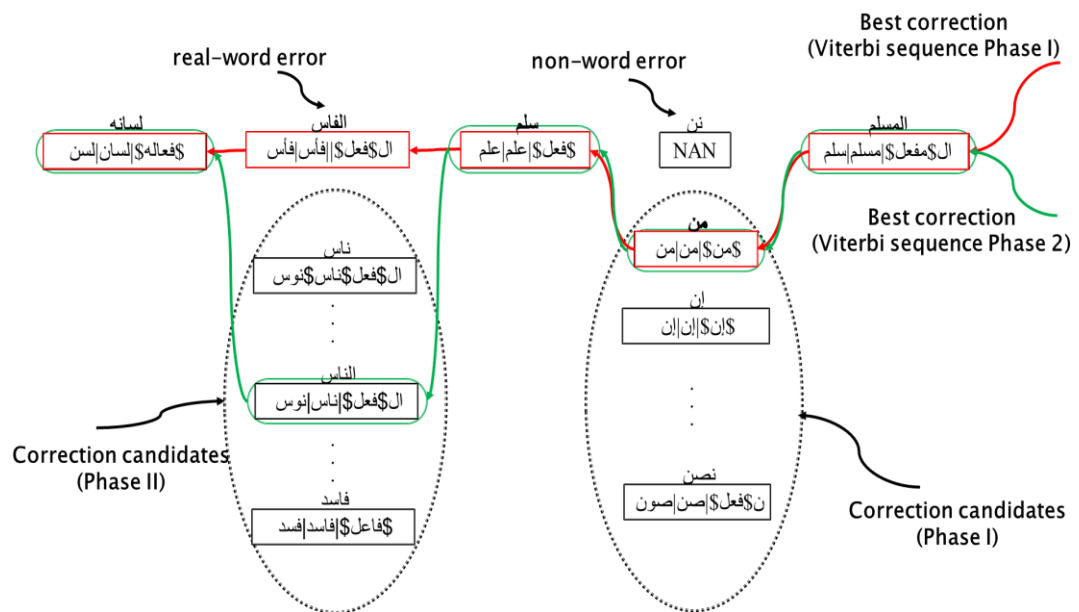


Figure 5.2 Examples of error detection and correction phases

5.3 System Formulation

Given (a) a set of words $W = \{w_1, w_2, \dots, w_m\}$ that represent Arabic words, (b) five finite sets of morphological sets $R = \{r_1, r_2, \dots, r_k\}$, $S = \{s_1, s_2, \dots, s_k\}$, $L = \{l_1, l_2, \dots, l_l\}$, $AP = \{a_1, a_2, \dots, a_h\}$ and $FP = \{p_1, p_2, \dots, p_h\}$ that represent the roots, stems, lemmas, affix patterns²⁰ and full patterns²¹ of Arabic words, respectively, (c) a morphological analysis function $M(w_i) = \{f_{i_1}, f_{i_2}, \dots, f_{i_{m_i}}\}$ that generate all possible analysis features tuples for any word w_i . Each analysis f_{i_j} is a tuple of 5 features $(r_j, s_j, l_j, a_j, p_j)$ where $r_j \in R$, $s_j \in S$, $l_j \in L$, $a_j \in AP$ and $p_j \in FP$, and (d) a candidate generation function $CCG(w) = \{w_1^c, w_2^c, \dots, w_m^c\}, w_i^c \in W, 1 \leq i \leq m$ that generates the probable correction candidates based on ECPD.

For any sentence $S = (w_1, w_2, \dots, w_k, \dots, w_{n-1}, w_n)$ with possible morphological features $F(S) = (M(w_1), M(w_2), \dots, M(w_k), \dots, M(w_{n-1}), M(w_n))$, we need to find the best correction sentence $S^* = (w_1^c, w_2^c, \dots, w_k^c, \dots, w_{n-1}^c, w_n^c)$, where

$w_i, 1 \leq i \leq n$ represent the sentence words;

$M(w_i), 1 \leq i \leq n$ represent the possible features' tuples of the word w_i ;

w_i^c represents the best correction of the word w_i ;

$$w_i^c, 1 \leq i \leq n \in \begin{cases} \{w_i \cup CCG(w_i)\} & \text{if } w_i \text{ was suspected as error} \\ \{w_i\} & \text{others} \end{cases}$$

²⁰ By affix pattern, we mean the original word pattern along with affixes separated by \$, for example the affix pattern for the word 'المهندون' is '\$المفتعل\$ون'

²¹ By full pattern, we mean the original word pattern along with affixes, for example the full pattern for the word 'المهندون' is 'المفتعلون'

The goal S^* can be achieved by finding the most likely sequence of features $F^* = (f_1^*, f_2^*, \dots, f_k^*, \dots, f_{n-1}^*, f_n^*)$ that most likely represent S^* where, f_i^* represent the best morphological features tuple of w_i^c .

By assuming that the probability of any word features depends only on the features that precede it (Markov assumption)²², and the probability of the next word depends only on its features (Markov output independence assumption)²³, we can define:

$$F^* = \text{argMax}(\prod_{i=1}^n p(w_i^c | f_i) \times \prod_{i=1}^n p(f_i | f_{i-1})) \quad (1)$$

The equation in (1) represents a maximization problem that can be solved using Viterbi algorithm. The algorithm was adopted to select the word in the correction candidates with the highest morphological features' probability in the sequence as the best correction. The behavior of the algorithm highly depends on the results of the error model (ECPG). For example, if the error model provide a space deletion correction among the generated candidates, the algorithm needs to consider computing two extra transitions and emissions in the position of the word. On the other hand, if the error model suggest the deletion of the suspected word, the algorithm needs to consider skipping of the current word to the next one. All these behaviors were maintained by the correction algorithm with the help of the error model. The resulting general markov correction model is shown in Figure 5.3

²² $p(t_k | t_1 \dots t_{k-1}) = p(t_k | t_{k-1})$, this follows that $p(t_1 \dots t_k) = \prod_{i=1}^k p(t_i | t_{i-1})$

²³ $p(w_k | t_k, w_{k-1}, t_{k-1}, \dots w_1, t_1) = p(w_k | t_k)$, this follow that $p(w_1 \dots w_n | t_1 \dots t_n) = \prod_{i=1}^n p(w_i | t_i)$

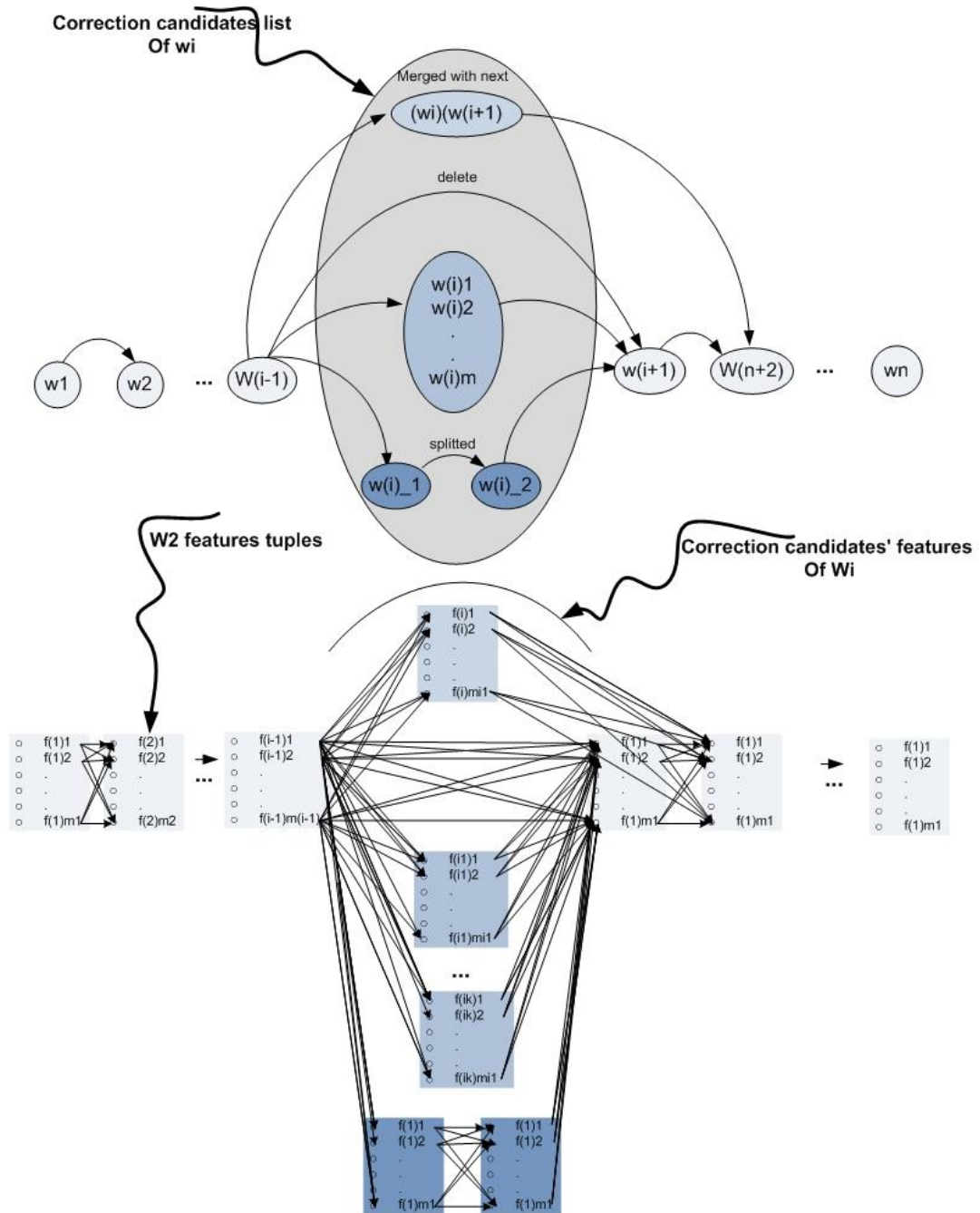


Figure 5.3 Spelling correction hmm model

According to the above formulations, the word w_i is considered as a spelling mistake if:

1. $M(w_i) = \emptyset$ (in the case of non-word error)
2. The probability of the sequence $p(f_i, f^* | f_1, f_2, \dots, f_{i-1})$ and $p(f_i, f^* | f_{i+1}, f_{i+2}, \dots, f_n)$ is less than a threshold θ (in the case of real-word error).

This means that any word will be considered as a spelling mistake if the word has no morphological analysis or if the morphological feature(s) of another word in the correction candidates has higher likelihood of fitting into the same context.

5.4 Estimation of Parameters

A supervised learning module was built for estimating the HMM parameters, viz., the transition probabilities $Prob\langle f_i | f_{i-1} \rangle$ and the emission probabilities $Prob\langle w_i | f_i \rangle$. The probabilities were estimated from an already-tagged corpus, using the maximum likelihood estimation method, in two phases. In the first phase, the probabilities (emission and transitions probabilities) for each feature (root, stem, lemma, affix pattern or full pattern) are estimated separately; zero-probabilities were smoothed using add- λ smoothing with backoff method (Christopher and Schuetze 1999). λ was set higher for words or features that rarely occur, since the training data may not contain rarely-occurring words and/or features in the language. In the second phase, the total smoothed probabilities of the whole feature tuple are then estimated as the product of the probabilities in the first phase, as shown in Equations 11 and 12. This is done under the assumption that the probability given a specific feature is independent from the probability given any other feature. This provides a customized estimation of the parameters for any feature or combinations of features.

The overall smoothed transition probabilities $Prob\langle f_i | f_{i-1} \rangle$ is defined as:

$$Prob(f_i | f_{i-1}) = \prod_{j=1}^m \frac{c(f_{(i-1)j}, f_{ij}) + \lambda_j \cdot Prob_{ff_backof}(f_{ij} | f_{(i-1)j})}{c(f_{(i-1)j}) + \lambda_j} \quad (11)$$

Where

f_{ij} represent the j^{th} morphological feature of the features tuple f_i

$c(f_{(i-1)j}, f_{ij})$ is the number of times the feature $f_{(i-1)j}$ appears in the training corpus followed by the feature f_{ij}

$c(f_{(i-1)j})$ is the number of times the feature $f_{(i-1)j}$ appears in the training corpus

λ_j is the number of j^{th} feature types such that $c(f_{(i-1)j}, f_{ij}) = 1$

The overall emission probabilities $Prob\langle f_i | f_{i-1} \rangle$ is defined as:

$$Prob(w_i | f_i) = \prod_{j=1}^m \frac{c(f_{ij}, w_i) + \lambda_j \cdot Prob_{wf_backoff}(w_i | f_{ij})}{c(f_{ij}) + \lambda_j} \quad (12)$$

where

$c(f_{ij}, w_i)$ is the number of times the word w_i appears in the training corpus with the feature f_{ij}

λ_j is the number of the word types such that $c(f_{ij}, w_i) = 1$

In Equations 11 and 12, m represents the number of features considered in the features tuple, where the maximum in our case is equal to 5, viz., root, stem, lemma, affix pattern and full pattern. λ_j was assigned a very small number, e.g. $(1e - 10)$, in the case when the number of singletons is zero ($\lambda_j = 0$). Since the sizes of the sets of roots, stems, lemmas, etc. vary greatly, the best value of λ_j in each case is determined during the system validation process as described in the next section.

The backoff estimates for each feature are defined as follows:

$$Prob_{ff_backoff}(f_{ij}|f_{(i-1)j}) = \frac{c(f_{ij})+1}{n_j} \quad (13)$$

$$Prob_{wf_backoff}(w_i|f_{ij}) = \frac{c(w_i) + 1}{n_j + V} \quad (14)$$

Where n_j denotes the number of j^{th} feature that were observed in the training data, and V denote the number of words that were observed in the training data. The above backoff estimate uses add-one smoothing; Some words that does not occur in the train data may happen to exist in the test data as novel words; these words are treated as if they had been replaced in the input by a single special word, out of vocabulary word, denoted by OOV, which is added to the set V . On the other hand, some features that does not occur in the train data may be generated by the morphological analyzer in the first phase as novel features; these features are treated as if they had been replaced by a single special feature that we call out of state, denoted by OOS, which is added to the value n_j , since the features are suggested by the morphological analyzer, not from the training data features.

5.5 Experimental Setup and Results

The general spelling detection and correction was evaluated using a manually annotated corpora, QALB²⁴ and KFUPM corpora. QALB annotated corpus does not provide any distinction between non-word and real-word errors. It only provides the type of the operation that generated the error (e.g. edit, merge, split, add before, add after, etc.). This required us to re-annotate the corpus in order to use it in our work. We handled a manual revision for the corpus after a preprocessing step. The preprocessing step was carried out with the help of Alkhalil, SWAM, Aramorph and the dictionary. Any word suggested to be a non-word error by all the aforementioned methods is annotated as non-word. On the other hand, any word suggested to be a real-word error by all these methods is annotated as real-word. Otherwise, if there is any conflict between the methods, the word is manually checked and annotated.

5.5.1 Handling Non-word Errors

The experiments concentrated on the ability of the system to detect and correct non-word errors compared to the other techniques. In order to have a good evaluation, three types of experiments were maintained. The first set of experiments concentrated on the problem of detecting non-word errors. The second set of experiments targeted the correction process, under the assumption of correct detection of errors. In the third set of experiments, detection and correction of Arabic text has been tested.

The results of non-word detection are shown in Table 5-1 using different corpora. SWAM non-word errors detection achieved around 65% F1-measure in the case of QALB corpus,

²⁴ QALB is a manually annotated errors corpus consisting of user comments collected from Al-Jazeera News webpage. It is mainly written in Modern Standard Arabic with almost one Million words with 243,075 errors.

and around 45% in the case of KFUPM corpus. It is clear from the results that SWAM non-word errors detection accuracies are lower as compared to Aramorph. The main reason for this is related to the lexicon lists used by SWAM. However, the results can be improved if SWAM's lexicon lists have been revised by an Arabic expert.

Table 5-1 Non-word error detection results

Corpus Name	Detection Method	Total errors	Detection		
			Recall	Precision	F1
QALB	SWAM	6205	53.17	86.50	65.85
	Aramorph		78.97	85.16	81.95
	Dictionary		52.94	64.32	58.08
	charNgrams		27.54	22.20	24.58
KFUPM	SWAM	90215	56.39	37.85	45.30
	Aramorph		66.65	44.26	53.19
	Dictionary		35.40	17.50	23.42
	charNgrams		24.32	4.71	7.88

In the case of correction process using HMM, the HMM model was built using a manually annotated corpus, NEMLAR²⁵ based on different features. The features are Root, Stem, Lemma, AffixSPattern, AffixOPattern, FSPattern and FOPattern. In order to determine the best HMM model features and the best window size, we used random set of sentences as validation set and maintained different experiments with different window sizes and features. The experiments employ the error patterns method for generation of candidates.

Table 5-2 Sample experiments with different window size and different model features

Model Features	Window Size	Top1			Top5			Top10		
		R	P	F1	R	P	F1	R	P	F1
R&S&L	3	5.21	5.95	5.56	39.58	45.24	42.22	50.00	57.14	53.33
R&S	3	4.17	4.88	4.49	39.58	46.34	42.70	58.33	68.29	62.92
R&S&ASP&AOP	3	15.62	16.67	16.13	51.04	54.44	52.69	71.88	76.67	74.19
R&S&FSP&FOP	3	6.25	7.32	6.74	52.08	60.98	56.18	72.92	85.37	78.65
R&S&L&ASP&AOP	3	17.71	19.32	18.48	50.00	54.55	52.17	67.71	73.86	70.65
R&S&L&ASP&AOP	3	7.29	8.54	7.87	54.17	63.41	58.43	66.67	78.05	71.91

²⁵ Manually annotated words from journalistic Arabic texts of different categories. Each word in the corpus was annotated with its prefix, root, pattern and suffix. The corpus has almost 500K words

S&ASP&AOP	3	13.54	14.44	13.98	51.04	54.44	52.69	73.96	78.89	76.34
S&FSP&FOP	3	7.29	8.43	7.82	52.08	60.24	55.87	73.96	85.54	79.33
S&L&ASP&AOP	3	17.71	19.32	18.48	53.12	57.95	55.43	67.71	73.86	70.65
R	5	3.12	3.66	3.37	27.08	31.71	29.21	41.67	48.78	44.94
R& FSP&FOP	5	9.38	10.84	10.06	38.54	44.58	41.34	62.50	72.29	67.04
R&S	5	4.17	4.88	4.49	40.62	47.56	43.82	56.25	65.85	60.67
R&S&ASP&AOP	5	15.62	16.85	16.22	48.96	52.81	50.81	67.71	73.03	70.27
R&S&L	5	6.25	7.32	6.74	32.29	37.80	34.83	50.00	58.54	53.93
R&S&L&ASP&AOP	5	16.67	18.18	17.39	50.00	54.55	52.17	63.54	69.32	66.30
R&S&L&FSP&FOP	5	6.25	7.32	6.74	53.12	62.20	57.30	64.58	75.61	69.66
S&FSP&FOP	5	7.29	8.43	7.82	47.92	55.42	51.40	73.96	85.54	79.33
R	7	3.12	3.66	3.37	29.17	34.15	31.46	42.71	50.00	46.07
R&ASP&AOP	7	12.50	13.48	12.97	40.62	43.82	42.16	62.50	67.42	64.86
R&FSP&FOP	7	7.29	8.54	7.87	38.54	45.12	41.57	60.42	70.73	65.17
R&S	7	4.17	4.88	4.49	40.62	47.56	43.82	56.25	65.85	60.67
R&S&ASP&AOP	7	14.58	15.56	15.05	52.08	55.56	53.76	69.79	74.44	72.04
R&S&L&ASP&AOP	7	17.71	19.32	18.48	50.00	54.55	52.17	64.58	70.45	67.39
R	9	3.12	3.61	3.35	22.92	26.51	24.58	39.58	45.78	42.46
R&S&ASP&AOP	9	15.62	16.67	16.13	50.00	53.33	51.61	70.83	75.56	73.12

Based on the experiments in Table 5-2 the next experiments are carried out using a window of size 3 and the features of Root, Stem, Lemma, AffixSPattern and AffixOPattern.

In order to have a clear indication about the correction accuracies, experiments were maintained with the assumption that the detection results are 100%. The non-word errors were manually tagged as suspected errors. Table 5-3 shows the results of the correction process of HMM compared to other techniques.

Table 5-3 Non-word error correction results (100% Detection)

Corpus	Correction Method	Candidate Generation Method	Total	Correction		
				Top1	Top5	Top10
				P	P	P
QALB	HMM	EP	6205	18.66	47.33	60.34
	HMM	ED		27.16	49.46	59.27
	NGRAMS	EP		30.51	63.38	69.14
	NGRAMS	ED		31.54	61.06	64.61
	HMM&Ngrams	EP		37.20	68.80	73.15
	HMM&Ngrams	ED		43.55	63.88	65.71
KFUPM	HMM	EP	90215	17.84	54.34	70.14
	HMM	ED		34.37	62.18	72.19
	NGRAMS	EP		37.78	75.09	79.09
	NGRAMS	ED		42.04	74.66	76.96
	HMM&Ngrams	EP		45.36	77.94	81.52
	HMM&Ngrams	ED		56.64	76.53	77.98

For testing the detection and correction accuracies, another experiments were conducted using QALB and KFUPM corpora. Table 5-4 shows the experimental results comparing the different used techniques.

Table 5-4 Non-word error correction results

Corpus	Detection Method	Correction Method	Cand. Gen. Method	Total	Correction								
					Top1			Top5			Top10		
					R	P	F1	R	P	F1	R	P	F1
QALB	SWAM	HMM	EP	6205	10.51	17.09	13.02	25.64	41.71	31.76	31.89	51.89	39.50
	SWAM	HMM	ED		15.33	24.93	18.98	26.51	43.13	32.84	30.57	49.74	37.87
	SWAM	NGRAMS	EP		15.65	25.46	19.38	32.70	53.20	40.50	35.39	57.58	43.84
	SWAM	NGRAMS	ED		16.66	27.11	20.64	32.34	52.62	40.06	33.46	54.43	41.44
	Aramorph	NGRAMS	EP		23.35	25.18	24.23	51.25	55.27	53.18	55.92	60.31	58.03
	Aramorph	NGRAMS	ED		24.38	26.29	25.30	49.91	53.82	51.79	52.86	57.00	54.85
	SWAM	HMM & NGrams	EP		19.39	31.54	24.01	35.20	57.26	43.60	37.47	60.96	46.41
	SWAM	HMM & NGrams	ED		23.06	37.52	28.57	33.07	53.80	40.96	33.67	54.77	41.70
KFUPM	SWAM	HMM	EP	90215	11.41	7.66	9.17	33.43	22.44	26.86	41.78	28.04	33.56
	SWAM	HMM	ED		21.81	14.64	17.52	38.25	25.67	30.72	42.63	28.62	34.25
	SWAM	NGRAMS	EP		22.40	15.04	18.00	43.91	29.47	35.27	46.25	31.05	37.15
	SWAM	NGRAMS	ED		24.53	16.47	19.71	43.65	29.30	35.07	44.55	29.90	35.79
	Aramorph	NGRAMS	EP		29.56	19.35	23.39	57.79	37.83	45.73	61.03	39.95	48.29
	Aramorph	NGRAMS	ED		32.75	21.44	25.91	58.01	37.97	45.90	59.63	39.03	47.18
	SWAM	HMM & NGrams	EP		27.90	18.72	22.41	45.53	30.56	36.57	47.70	32.02	38.32
	SWAM	HMM & NGrams	ED		21.85	14.67	17.55	38.25	25.67	30.72	42.67	28.64	34.28

The results in Table 5-3 and Table 5-4 are relatively low compared to other systems. This is due to several factors. The morphological analysis lexicon lists need more revision for improving the detection results, as mentioned earlier. In addition, a larger corpus should be used by the error model to improve the coverage of candidates' generation. Finally, a larger annotated corpus should be used by the HMM model to improve the correction accuracy.

5.5.2 Handling Real-word Errors

The task of real-word error detection was not easy to handle using the suggested model and the existing data. The main problem was determining the threshold that should be used for the detection of real-word errors. A set of experiments were developed to extract the threshold ranges to be used for the task of detection of the real-word errors. The experiments were mainly based on QALB corpus. A set of 350 sentences were selected, each having a single real-word error within a window size of 11. Moreover, two other sets of sentences were generated from the selected sets. The first set contains error free sentences, where each error word (real-word or non-word) was replaced with its correction. The second set of sentences contain real word errors only. This was achieved by replacing any non-word error in these sentences with their correction. The distributions of forward, backward and forward-backward probabilities for each set were generated. Sample results of the probabilities distribution are shown in Table 5-5. The remaining results are provided in Appendix C. It is clear that there are overlapping areas between the probabilities of the real-word errors of the selected set (Red color) and the two generated sets: error-free set (Blue color) and only real-word error set (Green color). This overlapping introduces a problem in the decision of whether a detected error is considered real-word or not.

Table 5-5 Real-word error probabilities distribution

W	Features	Forward	Backward	Forward-Backward
3	R	<p>Histogram of multiple variables R30 sta 45v*10000c</p> <p>W_F1_Correct = 9812*10*normal(x, -10.2548, 10.2018) W_F1_SingleE_Real = 372*10*normal(x, -22.8534, 15.79) W_F1_SingleE_Correct = 9441*10*normal(x, -10.165, 10.2025)</p> <p>No of obs</p> <p>W_F1_Correct W_F1_SingleE_Real W_F1_SingleE_Correct</p>	<p>Histogram of multiple variables R30 sta 45v*10000c</p> <p>W_B1_Correct = 9812*10*normal(x, -9.6816, 10.3455) W_B1_SingleE_Real = 372*10*normal(x, -11.0684, 12.3437) W_B1_SingleE_Correct = 9441*10*normal(x, -10.0215, 10.8326)</p> <p>No of obs</p> <p>W_B1_Correct W_B1_SingleE_Real W_B1_SingleE_Correct</p>	<p>Histogram of multiple variables R30 sta 45v*10000c</p> <p>W_FB1_Correct = 9812*20*normal(x, -11.9358, 15.9422) W_FB1_SingleE_Real = 372*20*normal(x, -16.2295, 19.34) W_FB1_SingleE_Correct = 9441*20*normal(x, -12.2711, 16.3356)</p> <p>No of obs</p> <p>W_FB1_Correct W_FB1_SingleE_Real W_FB1_SingleE_Correct</p>
3	R & ASP & AOP	<p>Histogram of multiple variables R10_ASP20_AOP30 sta 45v*10000c</p> <p>W_F1_Correct = 9812*50*normal(x, -75.7202, 67.5123) W_F1_SingleE_Real = 372*50*normal(x, -160.3408, 112.7227) W_F1_SingleE_Correct = 9441*50*normal(x, -76.741, 67.8784)</p> <p>No of obs</p> <p>W_F1_Correct W_F1_SingleE_Real W_F1_SingleE_Correct</p>	<p>Histogram of multiple variables R10_ASP20_AOP30 sta 45v*10000c</p> <p>W_B1_Correct = 9812*50*normal(x, -79.7914, 74.3127) W_B1_SingleE_Real = 372*50*normal(x, -95.7346, 74.3995) W_B1_SingleE_Correct = 9441*50*normal(x, -83.3346, 77.7333)</p> <p>No of obs</p> <p>W_B1_Correct W_B1_SingleE_Real W_B1_SingleE_Correct</p>	<p>Histogram of multiple variables R10_ASP20_AOP30 sta 45v*10000c</p> <p>W_FB1_Correct = 9812*50*normal(x, -113.7831, 92.1209) W_FB1_SingleE_Real = 372*50*normal(x, -172.3785, 110.4367) W_FB1_SingleE_Correct = 9441*50*normal(x, -117.928, 95.1429)</p> <p>No of obs</p> <p>W_FB1_Correct W_FB1_SingleE_Real W_FB1_SingleE_Correct</p>

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This chapter concludes and summarizes the main contributions and outcomes of this thesis. The main goal of this research is to support spell checking detection and correction for Arabic text. This chapter also discusses the main conclusions, the limitations, the possible enhancement and the future research directions

6.1 Conclusions

Spell checking detection and correction capabilities are vital in most state of the art text editing related applications. They are also important in correcting errors of Optical Character Recognition (OCR) output including offline and on-line text recognition systems. In this thesis, we designed and implemented a set of tools for supporting the task of error detection and correction. The developed tools are integrated into a single prototype system for error detection, correction candidates' generation and error correction.

A morphological analyzer that uses a Sliding Window Asynchronous Matching (SWAM) approach has been developed and extended to include an additional disambiguation process and bigrams compatibility checking for morphological analysis. SWAM is a lexicon driven approach that uses morphological derivational forms (window patterns) to extract the probable morphological feature tuples for any given input word. The morphological feature tuple includes the root, the stem, the lemma, the pattern, and the affixes. The original existing SWAM lexicon lists were not enough to run the experiments, as many mapped

roots and patterns were not present in the lists. This forced us to develop our own lists in order for SWAM to work. In this regards, we generated the required lists from Alkhalil morphological system. The generated lists were large and required an extensive revision, which we partially did. However, the lists still have some extra elements which negatively affect the results accuracy. Hence, the help of an expert in the Arabic language is vital for revising those lists and improving the system's performance. SWAM was also extended to perform affixes compatibility checking in order to reject all non-compatible patterns. Features disambiguation was handled using a Markovian based Viterbi algorithm. This morphological analysis and disambiguation system can be described as a root-based stemmer, lemmatizer, and morphological pattern extractor which can be used to serve different NLP applications. The morphological analyzer is used here to support the error model and the error detection process. The results of the morphological analyzer reported an accuracy of 97.13% for roots, 98.20% for stems and 95.80% for lemmas, based on NEMLAR corpus.

We designed and implemented a novel data driven error model that is based on the morphological patterns. The error model learns the types and forms of the language patterns from an already annotated corpora. SWAM was used in building the model. The model supports the candidate's generation and ranking task for any spelling correction system. Based on QALB and KFUPM corpora, the error model effectiveness was evaluated. The results show that the error model can support the correction process with almost 85% coverage; this ratio can be improved by including more corpora in the learning process. Moreover, the error model provides a simple way of analyzing the types of errors for any annotated corpora. It generates a set of reports that can provide an indication of the

complexity of corpus errors, which assists in attempting to better understand the types of generated errors.

A spell checking prototype that handles different error types was investigated. The spell checker was developed through an integration between the system components, the morphological analyzer, the error model, and the HMM model. The general spelling detection and correction was evaluated using a manually annotated corpora, viz., QALB²⁶ and KFUPM corpora. The results are relatively low compared to the other systems. This is due to several factors. The morphological analysis lexicon lists contain extra entries that affect the detection accuracy, as the lexicon lists are not completely verified. The error model does not provide the correct correction candidates for all the errors, it has a limited coverage. Moreover, the HMM model was trained with a limited size dataset.

6.2 Future Directions

The morphological analysis lexicon lists need more revision by experts in the Arabic language for improving the detection results. Also, a larger corpus should be used by the error model to improve the coverage of candidates generation and a larger dataset should be used by the HMM model to provide more accurate probabilities and improve the correction process.

The morphological analysis and disambiguation system can be extended by incorporating more functionalities. For example, the POS tagging feature can be integrated with SWAM existing features.

²⁶ QALB is a manually annotated errors corpus consisting of user comments collected from Al-Jazeera News webpage. It is mainly written in Modern Standard Arabic with almost one Million words with 243,075 errors.

The spell correction component of the spell checker can be investigated by applying features generated from another system, rather than SWAM. For example, Alkhalil system can be used for annotating any plain corpus and the generated features can then be used by the HMM model for the correction process.

Different smoothing techniques for HMM parameters estimation can be used. Such techniques may positively influence the model probabilities and hence may provide better disambiguation or correction accuracies.

References

- [1] Y. Hassan, M. Aly, and A. Atiya, "Arabic Spelling Correction using Supervised Learning," presented at the Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing (ANLP), Doha, Qatar, 2014.
- [2] T. Zerrouki, Alhawaity, K., & Balla, A. , "Autocorrection Of Arabic Common Errors For Large Text Corpus," presented at the In Proceedings of EMNLP Workshop on Arabic Natural Language Processing: QALB Shared Task, Doha, Qatar, 2014.
- [3] K. Kukich, "Technique for automatically correcting words in text," *ACM Computing Surveys* 1992.
- [4] A. Mahdi, "Spell Checking and Correction for Arabic Text Recognition," *Master's thesis, KFUPM University*, 2012.
- [5] M. Nejja and Y. Abdellah, "Correction of the Arabic derived words using surface patterns," presented at the 2014 5th Workshop on Codes, Cryptography and Communication Systems (WCCCS), El Jadida, Morocco, 2014.
- [6] M. Attia, Pecina, P., Samih, Y., Shaalan, K. F., & Van Genabith, J., "Improved Spelling Error Detection and Correction for Arabic.," presented at the Coling 2012, Mumbai, India, 2012.
- [7] M. Al-Jefri, "Real-word error detection and correction in Arabic text," *Master thesis, King Fahd University of Petroleum and minerals*, 2013.
- [8] M. Sawalha, "Open-source Resources and Standards for Arabic Word Structure Analysis.," *PhD, University of Leeds*, 2011.
- [9] J. Mayfield and P. McNamee, "Single n-gram stemming," presented at the Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval - SIGIR '03, Toronto, Canada, 2003.
- [10] I. A. Al-Sughaiyer and I. A. Al-Kharashi, "Arabic morphological analysis techniques: A comprehensive survey," *Journal of the American Society for Information Science and Technology*, vol. 55, 2004.
- [11] A. H. Aliwy, "Arabic Morphosyntactic Raw Text Part of Speech Tagging System.," *PhD , University of Warsaw*, 2013.
- [12] M. Ababneh, et al. , "Building an effective rule-based light stemmer for arabic language to improve search effectiveness.," *Int. Arab J. Inf. Technol. ,* 2012.
- [13] N. Y. Habash, "Introduction to Arabic Natural Language Processing," *Synthesis Lectures on Human Language Technologies*, vol. 3, pp. 1-187, 2010/01 2010.
- [14] T. Buckwalter, "Buckwalter {Arabic} Morphological Analyzer Version 1.0.," *Linguistic Data Consortium (LDC), University of Pennsylvania,2002. Catalog No:LDC2002L49.,* 2002.
- [15] T. Buckwalter, "Buckwalter Arabic Morphological Analyzer Version 2.0," *Linguistic Data Consortium (LDC), Philadelphia,2004. Catalog No:LDC2004L02.,* 2004.
- [16] A. Boudlal, A. Lakhouaja, A. Mazroui, A. Meziane, M. Bebah, and M. Shoul, "Alkhalil morpho sys1: A morphosyntactic analysis system for arabic texts," in *International Arab conference on information technology*, Benghazi Libya, 2010.

- [17] M. Altabba, A. Al-Zaraee, and M. A. Shukairy, "An Arabic morphological analyzer and part-of-speech tagger.," *A Thesis Presented to the Faculty of Informatics Engineering, Arab International University, Damascus, Syria*, 2010.
- [18] M. Sawalha, Eric Atwell, and Mohammad AM Abushariah., "SALMA: standard Arabic language morphological analysis.," *Communications, Signal Processing, and their Applications (ICCSPA)*, 2013.
- [19] M. A.-B. Arfath Pasha, Mona Diab, Ahmed El Kholy, Ramy Eskander, Nizar Habash, Manoj Pooleery, Owen Rambow, and Ryan M. Roth "Madamira: A fast, comprehensive tool for morphological analysis and disambiguation of Arabic.," *Proceedings of the Language Resources and Evaluation Conference (LREC), Reykjavik, Iceland.* , 2014.
- [20] I. Bounhas, R. Ayed, B. Elayeb, and N. Bellamine Ben Saoud, "A hybrid possibilistic approach for Arabic full morphological disambiguation," *Data & Knowledge Engineering*, vol. 100, 2015.
- [21] S. Khoja, "APT: Arabic Part-of-Speech Tagger. ," presented at the Proceedings of the Student Workshop at NAACL, Pittsburgh, PA, USA, 2001.
- [22] A. Boudlal, Bebah, M. O. A. O., Lakhouaja, A., Mazroui, A., & Meziane, A., "A Markovian approach for arabic root extraction.," *The International Arab Journal of Information Technology*, 2011.
- [23] Y. Attia. M., M., and Choukri., K. , "Specifications of the Arabic Written Corpus produced within the NEMLAR project;," <http://www.nemlar.org/>, 2005.
- [24] M. El-Defrawy, Y. El-Sonbaty, and N. A. Belal, "CBAS: Context Based Arabic Stemmer," *International Journal on Natural Language Computing (IJNLC)*, vol. 4, 2015.
- [25] S. Alansary, Magdy Nagi, and Noha Adly., "Building an international corpus of Arabic (ICA): Progress of compilation stage.," presented at the 7th International conference on language engineering, Egypt, 2007.
- [26] M. Hadni, A. Lachkar, and S. A. Ouatik, "A new and efficient stemming technique for Arabic Text Categorization," presented at the Multimedia Computing and Systems (ICMCS), 2012 International Conference on IEEE, Tangiers, Morocco, 2012.
- [27] L. Al-Sulaiti, and Eric Steven Atwell., "The design of a corpus of contemporary Arabic.," *International Journal of Corpus Linguistics*, 2006.
- [28] S. Deorowicz, and Marcin G. Ciura. , "Correcting spelling errors by modelling their causes.," *International journal of applied mathematics and computer science* 2005.
- [29] B. Hamza, et al., "For an Independent Spell-Checking System from the Arabic Language Vocabulary," *International Journal of Advanced Computer Science and Applications*, 2014.
- [30] W. Zaghouani, Taha Zerrouki, and Amar Balla. , " A Rule-Based Correction Method of Common Arabic Native and Non-Native Speakers' Errors.," presented at the In Proceedings of ACL Workshop on Arabic Natural Language Processing, Beijing, China, 2015.
- [31] G. Hicham, Y. Abdallah, and B. Mostapha, "Introduction of the weight edition errors in the Levenshtein distance," *International Journal of Advanced Research in Artificial Intelligence*, vol. 1, 2012.

- [32] E. Mays, Damerau, F. J., & Mercer, R. L. , "Context based spelling correction," *Information Processing & Management*, 1991.
- [33] A. Wilcox-O'Hearn, Hirst, G., & Budanitsky, A. , "Real-word spelling correction with trigrams: A reconsideration of the Mays, Damerau, and Mercer model.," *Computational Linguistics and Intelligent Text Processing*, 2008.
- [34] A. Islam, and Diana Inkpen. , "Real-word spelling correction using Google Web IT 3-grams," presented at the Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing Stroudsburg, PA, USA, 2009.
- [35] C. Ben Othmane Zribi, and Mohamed Ben Ahmed, "Detection of semantic errors in Arabic texts," *Artificial Intelligence*, vol. 1, 2013.
- [36] N. Tomeh, et al "A Pipeline Approach to Supervised Error Correction for the QALB-2014 Shared Task," presented at the In Proceedings of EMNLP Workshop on Arabic Natural Language Processing: QALB Shared Task, Doha, Qatar, 2014.
- [37] M. N. Nawar, & Ragheb, M. M. , "Fast and Robust Arabic Error Correction System," presented at the In Proceedings of EMNLP Workshop on Arabic Natural Language Processing: QALB Shared Task, Doha, Qatar, 2014.
- [38] K. Shaalan, Rana Aref, and Aly Fahmy., "An approach for analyzing and correcting spelling errors for non-native Arabic learners," presented at the Informatics and Systems (INFOS), The 7th International Conference on. IEEE, Cairo University, Egypt, 2010
- [39] K. F. Shaalan, Attia, M., Pecina, P., Samih, Y., & van Genabith, J., "Arabic Word Generation and Modelling for Spell Checking," presented at the The eighth international conference on Language Resources and Evaluation (LREC), istanbul, turkey, 2012.
- [40] A. Hassan, Sara Noeman, and Hany Hassan. , "Language Independent Text Correction using Finite State Automata," presented at the The Third International Joint Conference on Natural Language Processing (IJCNLP), Hyderabad, India, 2008.
- [41] B. Mohit, Rozovskaya, A., Habash, N., Zaghoulani, W., & Obeid, O. , "The first QALB shared task on automatic text correction for Arabic," presented at the In Proceedings of EMNLP Workshop on Arabic Natural Language Processing: QALB Shared Task, Doha, Qatar, 2014.
- [42] M. I. Alkanhal, M. A. Al-Badrashiny, M. M. Alghamdi, and A. O. Al-Qabbany, "Automatic stochastic Arabic spelling correction with emphasis on space insertions and deletions," *IEEE Transactions on Audio, Speech, and Language Processing*, 2012.

APPENDICES

Appendix A. The Details Results of Morphological Analysis and Disambiguation

Table 0-1 Result of morphological analysis using NEMLAR corpus

(Variable and Fixed words)

Without affixes compatibility

	Phase I	Phase II								
		AOP, ASP,S ,L ,R	Root	Stem	Lemma	R&FS P	R&FO P	S , L , R	ASP &R	R&S& FSP
Affix Original Pattern (AOP)	98.58	95.13	46.99	33.88	70.57	56.00	79.21	91.72	73.15	72.25
Affix Surface Pattern (ASP)	98.99	96.03	55.63	44.01	65.76	75.13	74.78	86.15	96.35	95.26
Full Original Pattern (FOP)	98.70	95.46	55.39	34.23	71.09	66.80	94.48	92.33	73.47	72.90
Full Surface Pattern (FSP)	99.08	96.74	72.16	44.89	74.96	96.95	89.21	87.38	96.88	96.89
Lemma (L)	98.98	95.80	55.24	33.54	94.30	64.04	79.69	94.63	73.21	72.78
Original Pattern (OP)	98.94	95.95	54.93	34.45	81.54	64.20	80.14	93.00	73.41	73.03

Root (R)	99.93	97.13	96.27	53.99	82.96	97.20	97.33	96.38	97.14	97.08
Root (R), Affix Original Pattern (AOP)	98.44	94.72	46.73	30.15	68.84	55.74	78.73	91.38	72.78	71.90
Root (R), Affix Surface Pattern (ASP)	97.98	94.83	55.21	38.12	63.71	74.19	74.31	85.38	94.96	93.90
Root (R), Full Original Pattern (FOP)	98.55	95.02	55.04	30.47	69.29	66.45	93.86	91.92	73.09	72.51
Root (R), Full Surface Pattern (FSP)	98.19	95.48	71.56	38.89	72.58	95.56	88.61	86.53	95.47	95.43
Root (R), Lemma (L)	98.97	95.57	55.06	30.89	81.13	63.82	79.59	94.35	72.91	72.53
Root (R), Lemma (L), Affix Original Pattern (AOP)	98.20	94.62	46.65	30.07	68.84	55.59	78.57	91.38	72.56	71.69
Root (R), Lemma (L), Full Original Pattern (FOP)	98.20	94.62	46.65	30.07	68.84	55.59	78.57	91.38	72.56	71.69
Root (R), Stem (S)	99.84	96.18	74.01	52.76	71.53	75.28	81.04	94.76	96.22	95.31
Root (R), Stem (S), Affix Original Pattern (AOP)	98.44	94.72	46.73	30.15	68.84	55.74	78.73	91.38	72.78	71.90
Root (R), Stem (S), Affix Surface Pattern (ASP)	97.98	94.83	55.21	38.12	63.71	74.19	74.31	85.38	94.96	93.90
Root (R), Stem (S), Full Original Pattern (FOP)	98.44	94.72	46.73	30.15	68.84	55.74	78.73	91.38	72.78	71.90
Root (R), Stem (S), Full Surface Pattern (FSP)	97.98	94.83	55.21	38.12	63.71	74.19	74.31	85.38	94.96	93.90
Root (R), Stem (S), Lemma (L)	98.80	95.01	47.23	30.53	70.49	55.75	78.92	93.53	72.70	71.90
Root (R), Stem (S), Lemma (L), Affix Surface Pattern (ASP), Affix Original Pattern (AOP)	97.70	94.11	44.41	28.36	63.18	55.33	73.39	84.81	72.22	71.37
Root (R), Stem (S), Lemma (L) , Full Surface Pattern (FSP), Full Original Pattern (FOP)	97.70	94.11	44.41	28.36	63.18	55.33	73.39	84.81	72.22	71.37

Stem (S)	99.95	98.20	75.90	97.16	85.19	76.84	82.73	96.93	98.45	97.37
Stem (S), Affix Original Pattern (AOP)	98.58	95.13	46.99	33.88	70.57	56.00	79.21	91.72	73.15	72.25
Stem (S), Affix Surface Pattern (ASP)	98.99	96.03	55.63	44.01	65.76	75.13	74.78	86.15	96.35	95.26
Stem (S), Full Original Pattern (FOP)	98.58	95.13	46.99	33.88	70.57	56.00	79.21	91.72	73.15	72.25
Stem (S), Full Surface Pattern (FSP)	98.99	96.03	55.63	44.01	65.76	75.13	74.78	86.15	96.35	95.26
Stem (S), Lemma (L)	98.84	95.22	47.39	33.12	82.20	55.93	79.02	93.78	72.98	72.07

Table 0-2 Result of morphological analysis using NEMLAR corpus

(Variable and Fixed words)

With affixes compatibility

	Phase II	Phase II								
		AOP, ASP,S ,L ,R	Root	Stem	Lemma	R&FS P	R&FO P	S , L , R	ASP &R	R&S& FSP
Affix Original Pattern (AOP)	98.57	95.20	63.13	44.70	76.55	69.72	90.03	93.22	76.46	76.59

Affix Surface Pattern (ASP)	98.77	96.07	72.28	53.16	71.42	87.44	84.74	87.49	96.49	95.88
Full Original Pattern (FOP)	98.63	95.52	65.68	44.96	76.92	72.65	94.60	93.60	76.72	76.92
Full Surface Pattern (FSP)	98.82	96.76	79.88	53.92	76.99	96.93	89.17	88.32	96.92	96.87
Lemma (L)	98.78	95.80	68.49	42.87	95.02	74.43	90.43	95.20	76.57	76.93
Original Pattern (OP)	98.81	95.98	68.38	45.41	83.73	74.63	90.95	94.26	76.80	77.27
Root (R)	99.66	96.93	96.62	58.18	84.25	96.98	97.16	96.50	96.96	96.89
Root (R), Affix Original Pattern (AOP)	98.44	94.79	62.69	39.74	74.68	69.37	89.48	92.80	76.04	76.20
Root (R), Affix Surface Pattern (ASP)	97.98	94.89	71.70	48.11	69.26	86.31	84.22	86.72	95.14	94.54
Root (R), Full Original Pattern (FOP)	98.49	95.08	65.22	39.97	75.01	72.27	93.99	93.13	76.29	76.51
Root (R), Full Surface Pattern (FSP)	98.05	95.51	79.22	48.80	74.65	95.55	88.59	87.47	95.54	95.45
Root (R), Lemma (L)	98.76	95.58	68.28	40.46	82.78	74.17	90.32	94.94	76.26	76.68
Root (R), Lemma (L), Affix Original Pattern (AOP)	98.20	94.69	62.58	39.62	74.63	69.22	89.32	92.70	75.82	75.99
Root (R), Lemma (L), Full Original Pattern (FOP)	98.20	94.69	62.58	39.62	74.63	69.22	89.32	92.70	75.82	75.99
Root (R), Stem (S)	99.53	96.03	85.74	57.24	76.85	87.34	91.72	95.41	96.26	95.78
Root (R), Stem (S), Affix Original Pattern (AOP)	98.44	94.79	62.69	39.74	74.68	69.37	89.48	92.80	76.04	76.20
Root (R), Stem (S), Affix Surface Pattern (ASP)	97.98	94.89	71.70	48.11	69.26	86.31	84.22	86.72	95.14	94.54
Root (R), Stem (S), Full Original Pattern (FOP)	98.44	94.79	62.69	39.74	74.68	69.37	89.48	92.80	76.04	76.20
Root (R), Stem (S), Full Surface Pattern (FSP)	97.98	94.89	71.70	48.11	69.26	86.31	84.22	86.72	95.14	94.54

Root (R), Stem (S), Lemma (L)	98.63	95.03	63.19	40.08	75.91	69.42	89.66	94.33	75.98	76.25
Root (R), Stem (S), Lemma (L), Affix Surface Pattern (ASP), Affix Original Pattern (AOP)	97.69	94.16	59.33	37.36	68.63	68.89	83.27	85.98	75.43	75.61
Root (R), Stem (S), Lemma (L) , Full Surface Pattern (FSP), Full Original Pattern (FOP)	97.69	94.16	59.33	37.36	68.63	68.89	83.27	85.98	75.43	75.61
Stem (S)	99.65	98.02	87.67	97.67	89.97	89.13	93.52	97.57	98.44	97.84
Stem (S), Affix Original Pattern (AOP)	98.57	95.20	63.13	44.70	76.55	69.72	90.03	93.22	76.46	76.59
Stem (S), Affix Surface Pattern (ASP)	98.77	96.07	72.28	53.16	71.42	87.44	84.74	87.49	96.49	95.88
Stem (S), Full Original Pattern (FOP)	98.57	95.20	63.13	44.70	76.55	69.72	90.03	93.22	76.46	76.59
Stem (S), Full Surface Pattern (FSP)	98.77	96.07	72.28	53.16	71.42	87.44	84.74	87.49	96.49	95.88
Stem (S), Lemma (L)	98.66	95.24	63.37	42.45	87.04	69.62	89.77	94.57	76.26	76.44

Appendix B. The Error Model Statistics Details

The generated error model analysis statistics are based on QALB manually annotated corpus, the statistics are as follows:

- A detailed corpus errors with their frequencies and the number of distinct corpus word errors is shown in Table 0-3. We represent the number of distinct corpus errors as the number of non-repeated word errors plus the number of repeated word errors. For example, almost 44% of the QALB corpus are a result of substitution error; more than 95% of the errors are a result of single letter substitution.
- Detailed corpus error patterns with their frequencies are shown in Table 0-4. The statistics include the ratios of the model generated error patterns based on the number of distinct errors in the corpus. This analysis gives us an indication of how effective the suggested error model will be in providing correction candidates for each types of spelling errors.

Table 0-3 QALB Annotated Corpus Errors Statistics

Change Type		# Changes	# Error Words								
			# Error Words (%)	Total	Non Repeated			Repeated			
					# (%)	Total (%)		# (%)	Total (%)		
Insertion	Space	1	9233(3.80)	13958(5.74)	19270(7.93)	3606(78.46)	6314	9025(83.64)	990(21.54)	1376	1765(16.36)
	Punctuation										
	Other Char		4725(1.94)			2708(87.52)			386(12.48)		
	Space	2	147(0.06)	914(0.38)		84(92.31)	702		7(7.69)	54	
	Other Char		572(0.24)			459(92.35)			38(7.65)		
	Mixed		195(0.08)			159(94.64)			9(5.36)		
	Space	3	40(0.02)	1454(0.60)		38(97.44)	212		1(2.56)	38	
	Other Char		82(0.03)			82(100.00)			0(0.00)		
	Mixed		1332(0.55)			92(71.32)			37(28.68)		
Any Char	>3	2944(1.21)		1797(85.82)		297(14.18)					
Deletion	Space	1	5184(2.13)	14237(5.86)	97668(40.18)	2492(82.35)	6849	33861(74.87)	534(17.65)	1632	11364(25.13)
	Punctuation		358(0.15)			0(0.00)			9(100.00)		
	Other Char		8695(3.58)			4357(80.00)			1089(20.00)		
	Space	2	79971(32.90)	81320(33.45)		24926(72.46)	25849		9472(27.54)	9585	
	Punctuation		1046(0.43)			670(87.35)			97(12.65)		
	Mixed		303(0.12)			253(94.05)			16(5.95)		
	Punctuation	3		962(0.40)			499			69	
	Other Char		473(0.19)			108(77.70)			31(22.30)		
	Mixed		489(0.20)			391(91.14)			38(8.86)		
Any Char	>3	1149(0.47)		664(89.49)		78(10.51)					
Substitution	Hamza	1	69321(28.52)	103058(42.40)	106702(43.90)	5723(61.30)	13113	14446(68.60)	3613(38.70)	6274	6611(31.40)
	Taamarbota		10263(4.22)			2154(61.37)			1356(38.63)		
	Yaa		6035(2.48)			739(66.28)			376(33.72)		
	Other		17439(7.17)			4497(82.88)			929(17.12)		
	HamzaHaa	2	1037(0.43)	3492(1.44)		257(69.46)	1218		113(30.54)	325	
	Others		2455(1.01)			961(81.93)			212(18.07)		
	Any Char	3	123(0.05)			89(89.00)			11(11.00)		
	Any Char	>3	29(0.01)			26(96.30)			1(3.70)		
	Transposition	Space	1	51(0.02)		448(0.18)	530(0.22)		47(95.92)	372	
Other		390(0.16)		324(94.46)	19(5.54)						
NoSpace Chars		2	23(0.01)	30(0.01)	23(100.00)	30		0(0.00)	0		
Mixed			7(0.00)		7(100.00)			0(0.00)			
NoSpace Chars		3	2(0.00)	20(0.01)	2(100.00)	20		0(0.00)	0		
Mixed			18(0.01)		18(100.00)			0(0.00)			
Any Char		>3	32(0.01)		32(100.00)			0(0.00)			

Mixed	Two Operations	2	3207(1.32)	16646(6.85)	18905(7.78)	2087(88.81)	8888	10467(87.30)	263(11.19)	1385	1523(12.70)			
		3	6085(2.50)			2692(84.79)			483(15.21)					
		>3	7354(3.03)			4109(86.54)			639(13.46)					
	Three Operations	3	331(0.14)	2171(0.89)		243(92.40)	1497		20(7.60)	136				
		4	420(0.17)			332			29					
		>4	1420(0.58)			922(91.38)			87(8.62)					
	Four Operations	4	7(0.00)	64		7(100.00)	63		0(0.00)	1				
		5	7(0.00)			7(100.00)			0(0.00)					
		>5	51(0.02)			49(98.00)			1(2.00)					
	No Change		23(0.01)			19(95.00)			1(5.00)					
	Totals		243075			68253 [76.23]			21283 [23.77]					

Table 0-4 The statistics of QALB distinct error patterns

Table 0-4 The statistics of QALB distinct error patterns														
Change Type			# Distinct Error Words (DEW)			# Error Patterns								
			# DEW			TEP: Total Error Patterns NREP Non-repeated EPattern REP Repeated EPattern								
						TEP (%)	NREP (%)	REP (%)	TEP (%)	NREP (%)	REP (%)	TEP (%)	NREP (%)	REP (%)
Insertion	Space	1	4596	7690	10790	1624 (35.34)	924 (20.10)	700 (15.23)	3906 (50.79)	2693 (35.02)	1213 (15.77)	6834 (63.34)	5184 (48.04)	1650 (15.29)
	Punctuation													
	Other Char		3094			2282 (73.76)	1769 (57.18)	513 (16.58)						
	Space	2	91	756		76 (83.52)	66 (72.53)	10 (10.99)	602 (79.63)	513 (67.86)	89 (11.77)			
	Other Char		497			367(73.84)	304 (61.17)	63 (12.68)						
	Mixed		168			159 (94.64)	143 (85.12)	16 (9.52)						
	Space	3	39	250		30 (76.92)	23 (58.97)	7 (17.95)	237 (94.80)	189 (75.60)	48 (19.20)			
	Other Char		82			80 (97.56)	78 (95.12)	2 (2.44)						
	Mixed		129			127 (98.45)	88 (68.22)	39 (30.23)						
	Any Char	>3	2094			2089 (99.76)	1789 (85.43)	300 (14.33)						
Deletion	Space	1	3026	8481	45225	1865 (61.63)	1449 (47.88)	416 (13.75)	4730 (55.77)	3455 (40.74)	1275 (15.03)	17786 (39.33)	11391 (25.19)	6395 (14.14)
	Punctuation		9			9 (100.00)	0 (0.00)	9 (100.00)						
	Other Char		5446			2856 (52.44)	2006 (36.83)	850 (15.61)						
	Space Punctuation	2	34398	35434		11249 (32.70)	6473 (18.82)	4776 (13.88)	11959 (33.75)	7032 (19.85)	4927 (13.90)			
	Other Char		767			461 (60.10)	337 (43.94)	124 (16.17)						
	Mixed		269			249 (92.57)	223 (82.90)	26 (9.67)						
	Punctuation	3		568					491 (86.44)	406 (71.48)	85 (14.96)			
	Other Char		139			124 (89.21)	88 (63.31)	36 (25.90)						
	Mixed		429			367 (85.55)	318 (74.13)	49(11.42)						
	Any Char	>3	742			606 (81.67)	497 (66.98)	109 (14.69)						
Substitution	Hamza	1	9336	19387	21057	3432 (36.74)	1773 (18.98)	1659 (17.76)	8545 (44.08)	5305 (27.36)	3240 (16.71)	9965 (47.32)	6414 (30.46)	3551 (16.86)
	Taamarbota		3510			648 (18.46)	285 (8.12)	363 (10.34)						
	Yaa		1115			450	209	241						

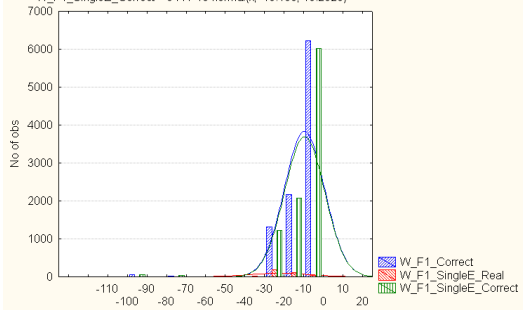
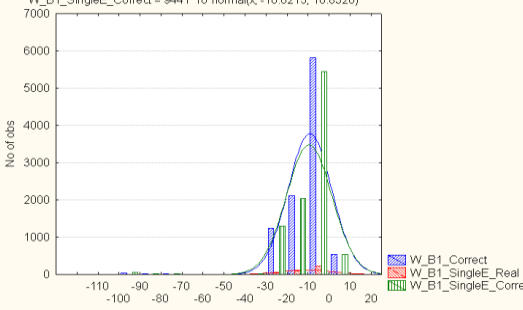
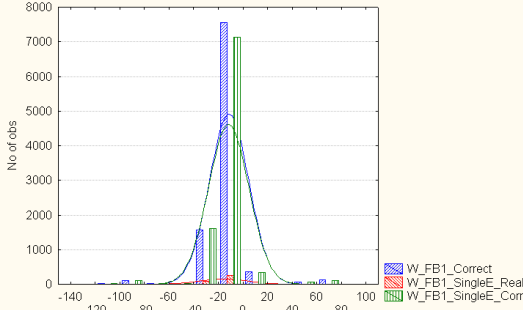
						(40.36)	(18.74)	(21.61)						
	Others		5426			4015 (74.00)	3038 (55.99)	977 (18.01)						
	HamzaHaa	2	370	1543		223 (60.27)	138 (37.30)	85 (22.97)	1293 (83.80)	994 (64.42)	299 (19.38)			
	Others		1173			1070 (91.22)	856 (72.98)	214 (18.24)						
	Any Char	3	100			100 (100.00)	89 (89.00)	11 (11.00)						
	Any Char	>3	27			27 (100.00)	26 (96.30)	1 (3.70)						
Transposition	Space	1	49	394	476	46 (93.88)	42 (85.71)	4 (8.16)	333 (84.52)	280 (71.07)	53 (13.45)	414 (86.97)	360 (75.63)	54 (11.34)
	Others		343			285 (83.09)	237 (69.10)	48 (13.99)						
	NoSpace Chars	2	23	30		23 (100.00)	23 (100.00)	0 (0.00)	30 (100.00)	30 (100.00)	0 (0.00)			
	Mixed		7			7 (100.00)	7 (100.00)	0 (0.00)						
	NoSpace Chars	3	2	20		2 (100.00)	2 (100.00)	0 (0.00)	19 (95.00)	18 (90.00)	1 (5.00)			
	Mixed		18			17 (94.44)	16 (88.89)	1 (5.56)						
	Any Char	>3	32			32 (100.00)	32 (100.00)	0 (0.00)						
Mixed	Two Operations ²⁷	2	2350	10273	11990	2104 (89.53)	1791 (76.21)	313 (13.32)	9434 (91.83)	7933 (77.22)	1501 (14.61)	11130 (92.83)	9477 (79.04)	1653 (13.79)
		3	3175			2657 (83.69)	2125 (66.93)	532 (16.76)						
		>3	4748			4673 (98.42)	4017 (84.60)	656 (13.82)						
	Three Operations	3	263	1633		250 (95.06)	222 (84.41)	28(10.65)	1613 (98.78)	1464 (89.65)	149 (9.12)			
		4	361			355 (98.34)	323 (89.47)	32 (8.86)						
		>4	1009			1008 (99.90)	919 (91.08)	89 (8.82)						
	Four Operations	4	7	64		7 (100.00)	7 (100.00)	0 (0.00)	64 (100.00)	63 (98.44)	1 (1.56)			
		5	7			7 (100.00)	7 (100.00)	0 (0.00)						
		>5	50			50	49	1						

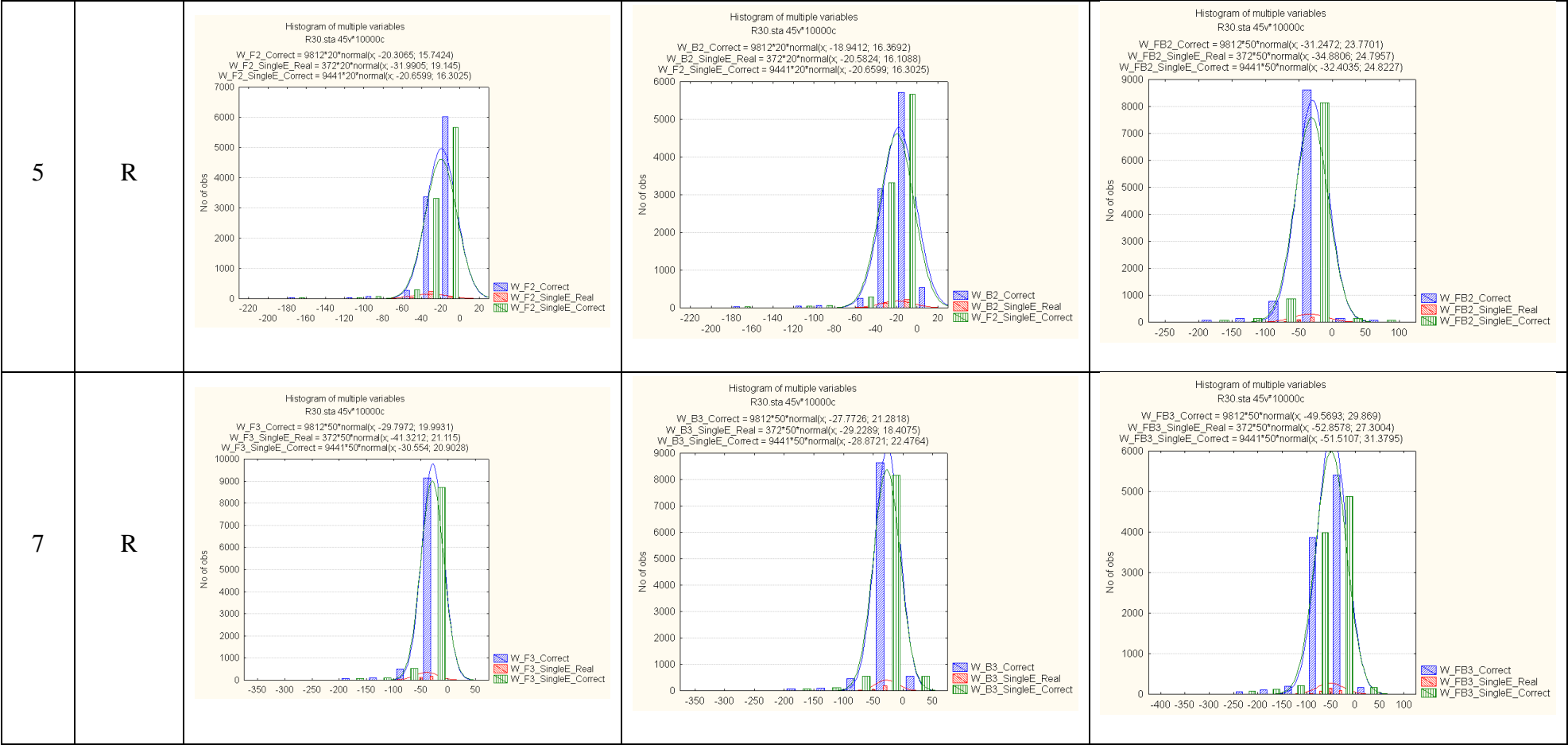
²⁷ Two operations means a mixing of any two change operations(Insertion, Deletion, Substitution or transposition)

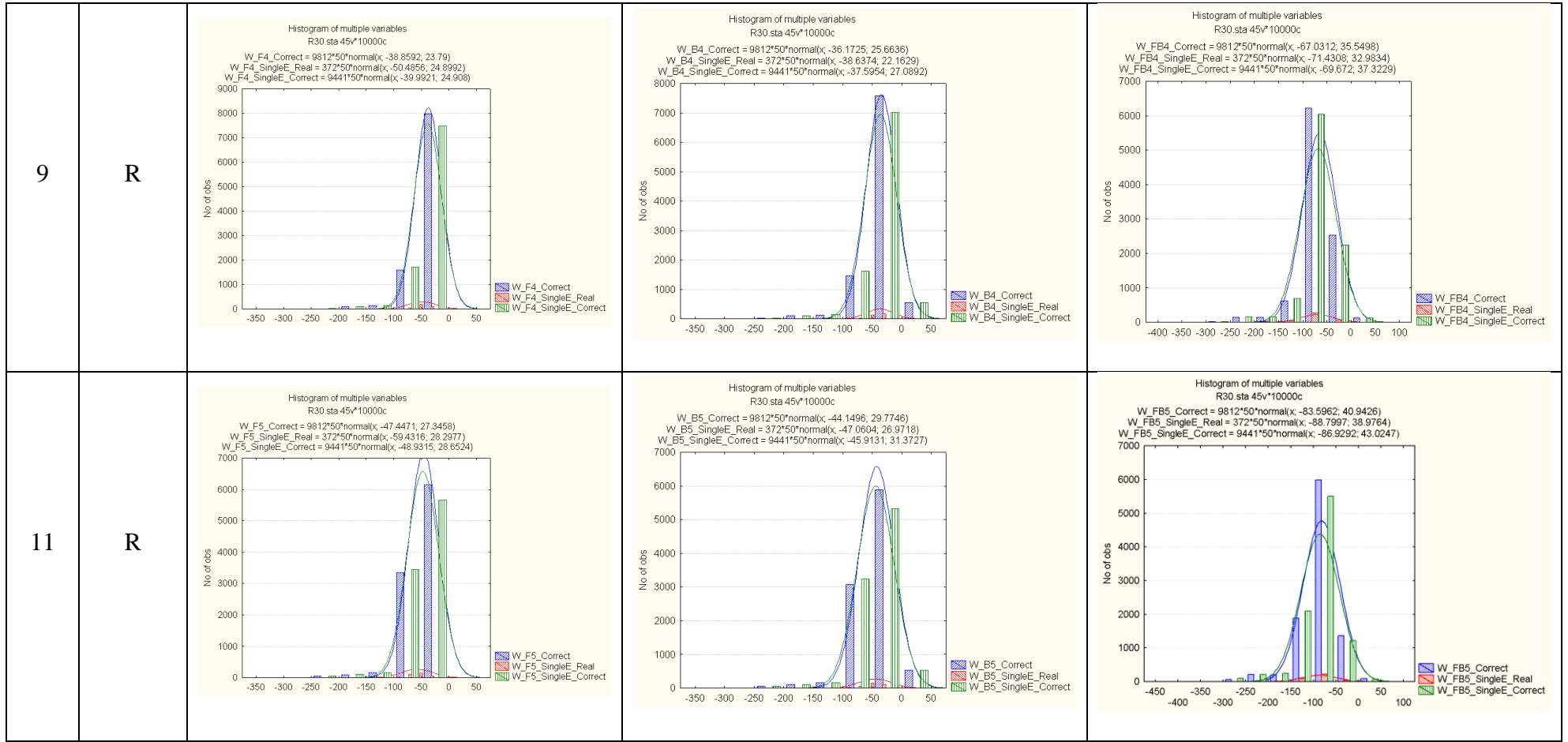
						(100.00)	(98.00)	(2.00)						
NO Change			20									19 (95.00)	17 (85.00)	2 (10.00)
Totals			89536 [36.83]									46129 [51.52]	32826 [36.66]	13303 [14.86]

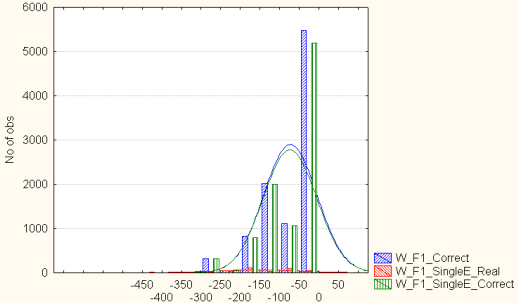
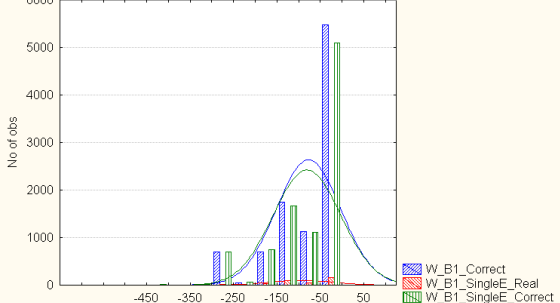
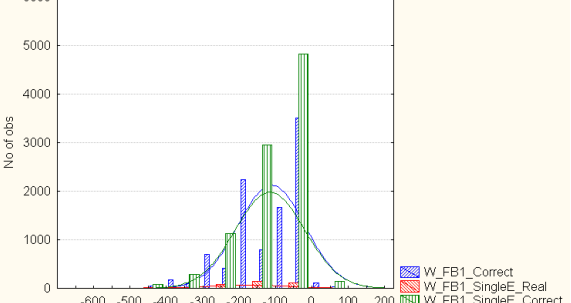
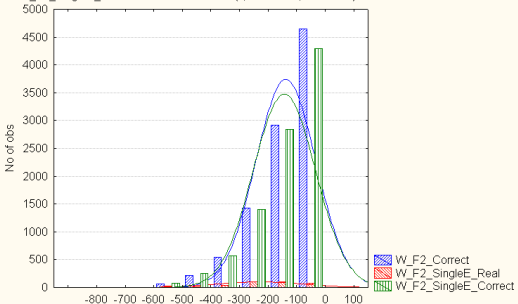
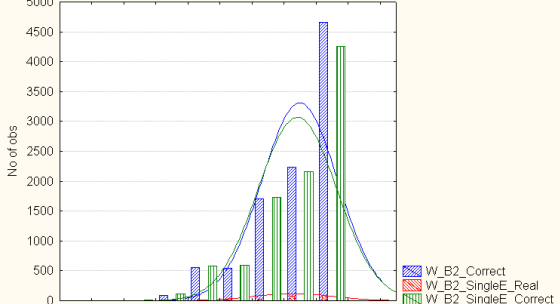
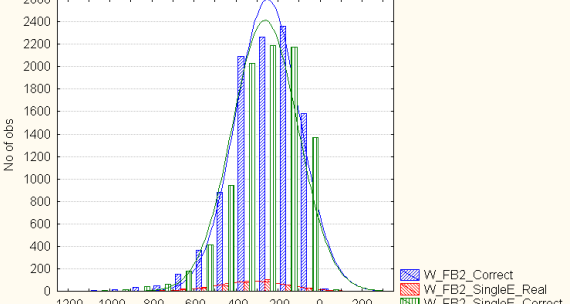
Appendix C. The GSpell Error Probability Distributions

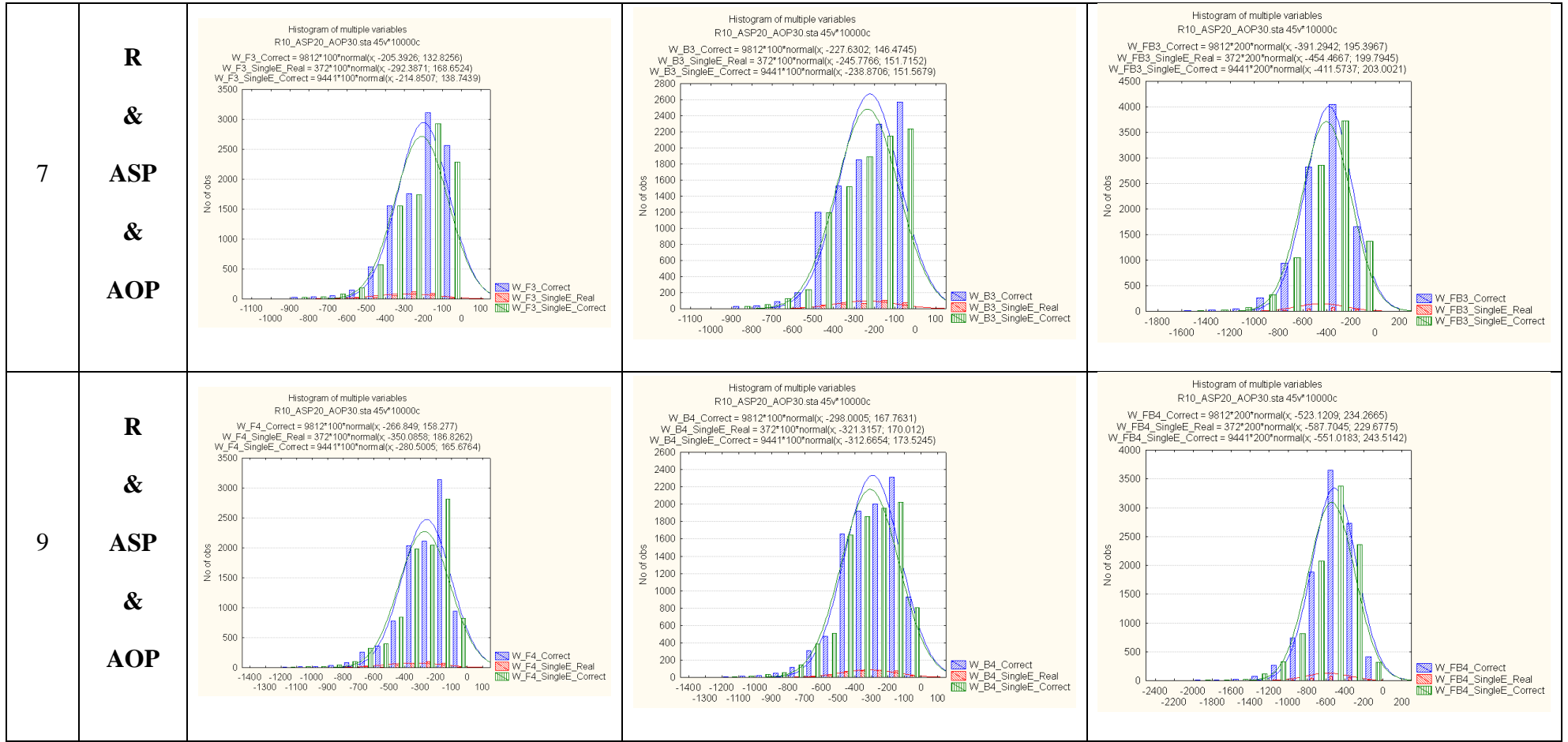
Table 0-5 Complete real-word error probabilities distribution

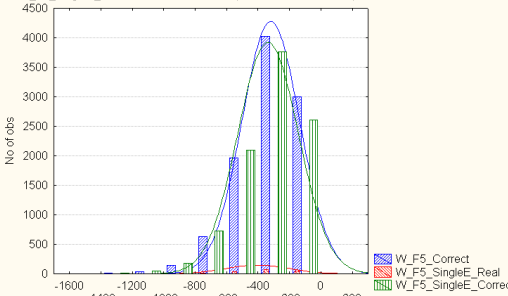
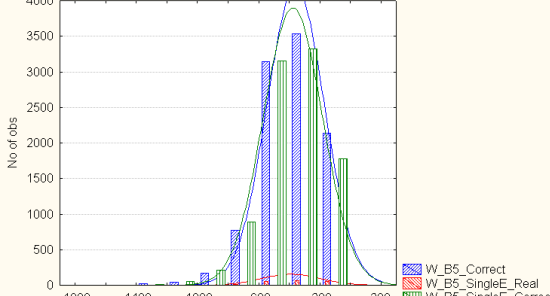
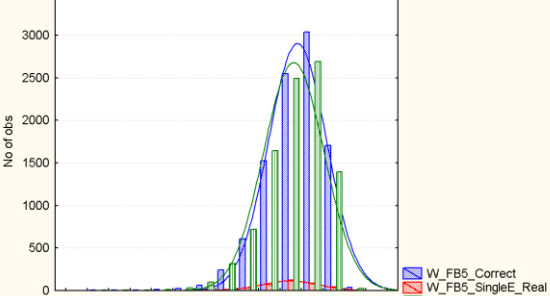
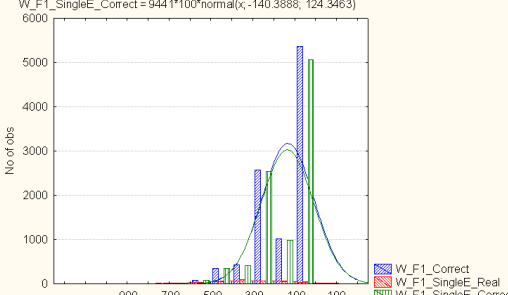
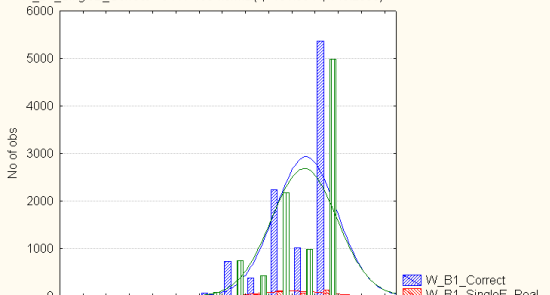
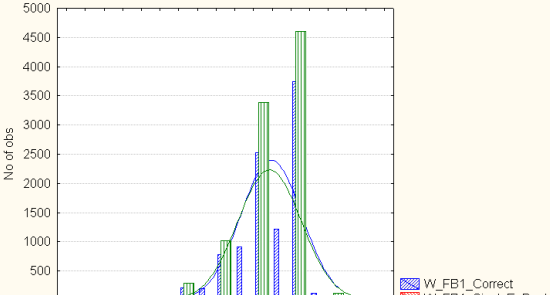
W	Features	Forward	Backward	Forward-Backward
3	R	<p>Histogram of multiple variables R30 sta 45v*10000c</p> <p>W_F1_Correct = 9812*10*normal(x, -10.2548, 10.2018) W_F1_SingleE_Real = 372*10*normal(x, -22.8534, 15.79) W_F1_SingleE_Correct = 944*10*normal(x, -10.185, 10.2025)</p>  <p>No of obs</p> <p>W_F1_Correct W_F1_SingleE_Real W_F1_SingleE_Correct</p>	<p>Histogram of multiple variables R30 sta 45v*10000c</p> <p>W_B1_Correct = 9812*10*normal(x, -9.6816, 10.3455) W_B1_SingleE_Real = 372*10*normal(x, -11.0684, 12.3437) W_B1_SingleE_Correct = 944*10*normal(x, -10.0215, 10.8326)</p>  <p>No of obs</p> <p>W_B1_Correct W_B1_SingleE_Real W_B1_SingleE_Correct</p>	<p>Histogram of multiple variables R30 sta 45v*10000c</p> <p>W_FB1_Correct = 9812*20*normal(x, -11.9358, 15.9422) W_FB1_SingleE_Real = 372*20*normal(x, -16.2295, 19.34) W_FB1_SingleE_Correct = 944*20*normal(x, -12.2711, 16.3356)</p>  <p>No of obs</p> <p>W_FB1_Correct W_FB1_SingleE_Real W_FB1_SingleE_Correct</p>

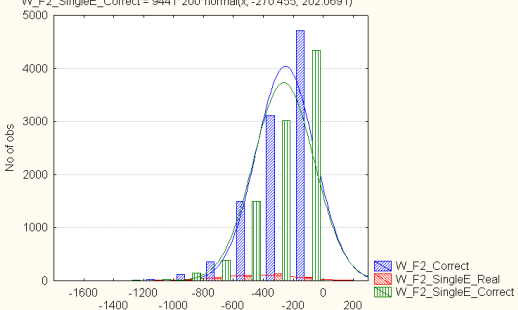
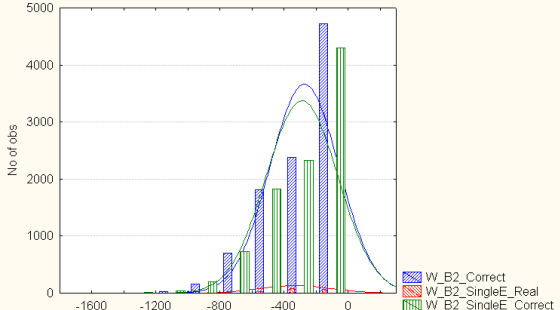
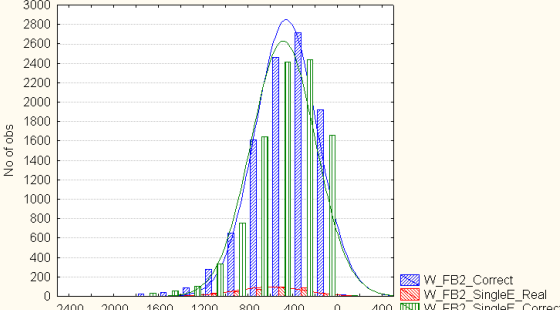
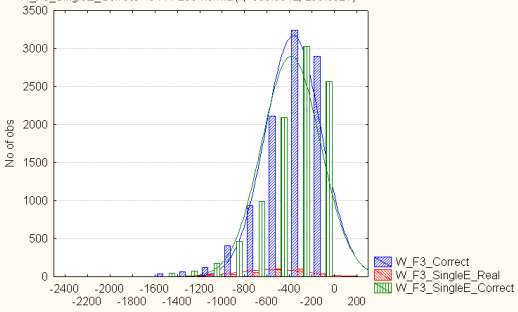
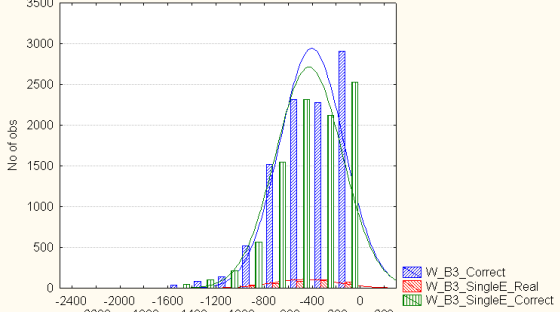
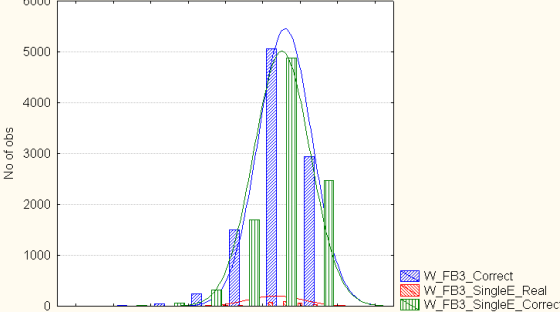


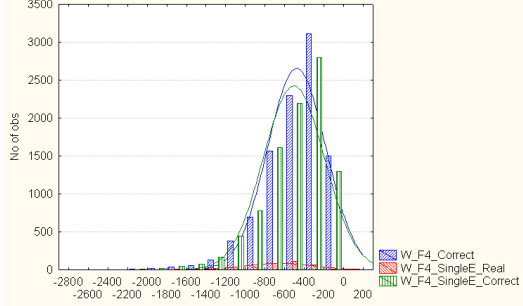
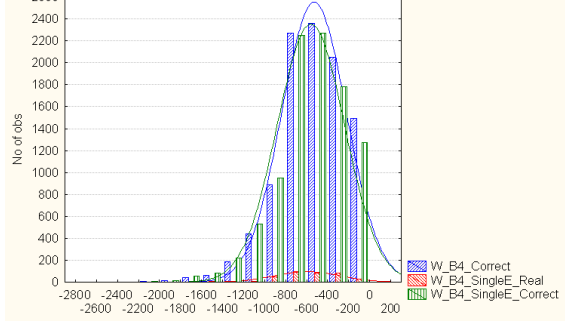
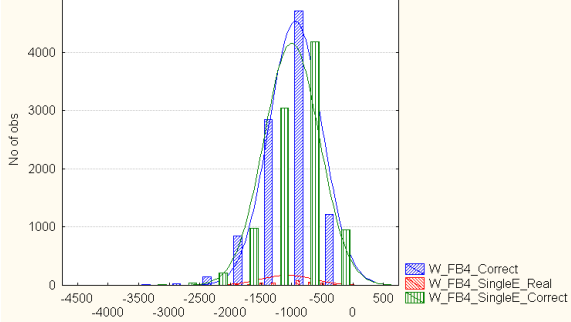


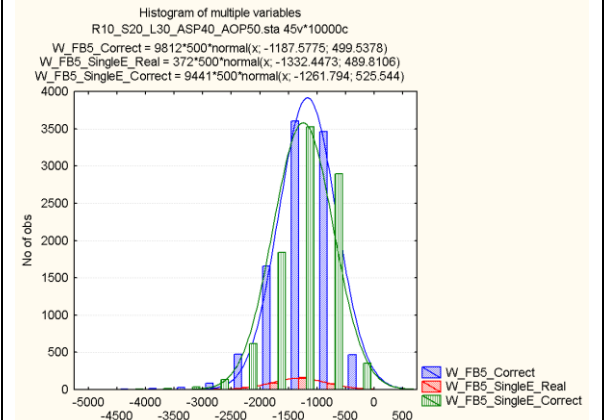
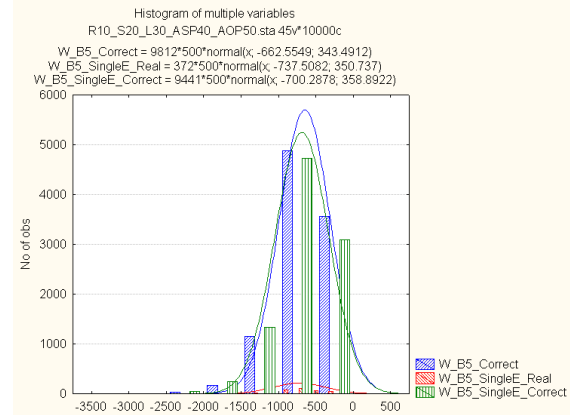
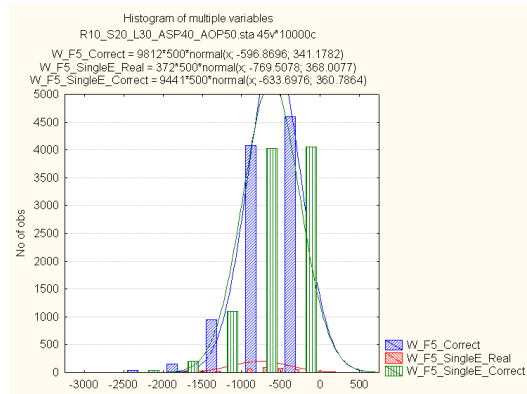
3	R & ASP & AOP	<p>Histogram of multiple variables R10_ASP20_AOP30 sta 45*10000c</p> <p>W_F1_Correct = 9812*50*normal(x, -75.7202, 67.5123) W_F1_SingleE_Real = 372*50*normal(x, -160.3408, 112.7227) W_F1_SingleE_Correct = 9441*50*normal(x, -76.741, 67.8784)</p> 	<p>Histogram of multiple variables R10_ASP20_AOP30 sta 45*10000c</p> <p>W_B1_Correct = 9812*50*normal(x, -79.7914, 74.3127) W_B1_SingleE_Real = 372*50*normal(x, -95.7346, 74.3995) W_B1_SingleE_Correct = 9441*50*normal(x, -83.3346, 77.7333)</p> 	<p>Histogram of multiple variables R10_ASP20_AOP30 sta 45*10000c</p> <p>W_FB1_Correct = 9812*50*normal(x, -113.7831, 92.1209) W_FB1_SingleE_Real = 372*50*normal(x, -172.3785, 110.4367) W_FB1_SingleE_Correct = 9441*50*normal(x, -117.928, 95.1429)</p> 
5	R & ASP & AOP	<p>Histogram of multiple variables R10_ASP20_AOP30 sta 45*10000c</p> <p>W_F2_Correct = 9812*100*normal(x, -141.7665, 104.6913) W_F2_SingleE_Real = 372*100*normal(x, -232.2282, 149.7872) W_F2_SingleE_Correct = 9441*100*normal(x, -146.8845, 108.4004)</p> 	<p>Histogram of multiple variables R10_ASP20_AOP30 sta 45*10000c</p> <p>W_B2_Correct = 9812*100*normal(x, -154.7479, 118.3253) W_B2_SingleE_Real = 372*100*normal(x, -174.3926, 124.3435) W_B2_SingleE_Correct = 9441*100*normal(x, -162.1054, 122.7406)</p> 	<p>Histogram of multiple variables R10_ASP20_AOP30 sta 45*10000c</p> <p>W_FB2_Correct = 9812*100*normal(x, -254.7858, 150.6895) W_FB2_SingleE_Real = 372*100*normal(x, -322.9238, 168.0704) W_FB2_SingleE_Correct = 9441*100*normal(x, -266.8423, 156.1813)</p> 



11	R & ASP & AOP	<p>Histogram of multiple variables R10_ASP20_AOP30.sta 45°10000c</p> <p>W_F5_Correct = 9812*200*normal(x, -325.6871; 183.0408) W_F5_SingleE_Real = 372*200*normal(x, -407.8343; 201.0389) W_F5_SingleE_Correct = 9441*200*normal(x, -343.2401; 192.0848)</p>  <p>No of obs</p>	<p>Histogram of multiple variables R10_ASP20_AOP30.sta 45°10000c</p> <p>W_B5_Correct = 9812*200*normal(x, -365.4476; 186.6637) W_B5_SingleE_Real = 372*200*normal(x, -394.9092; 190.3608) W_B5_SingleE_Correct = 9441*200*normal(x, -383.3517; 193.189)</p>  <p>No of obs</p>	<p>Histogram of multiple variables R10_ASP20_AOP30.sta 45°10000c</p> <p>W_FB5_Correct = 9812*200*normal(x, -649.4061; 269.8683) W_FB5_SingleE_Real = 372*200*normal(x, -719.1465; 261.4372) W_FB5_SingleE_Correct = 9441*200*normal(x, -684.4442; 281.1761)</p>  <p>No of obs</p>
3	R & S & L & ASP & AOP	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50.sta 45°10000c</p> <p>W_F1_Correct = 9812*100*normal(x, -138.03; 123.5005) W_F1_SingleE_Real = 372*100*normal(x, -310.9437; 203.3464) W_F1_SingleE_Correct = 9441*100*normal(x, -140.3888; 124.3463)</p>  <p>No of obs</p>	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50.sta 45°10000c</p> <p>W_B1_Correct = 9812*100*normal(x, -144.5036; 133.5463) W_B1_SingleE_Real = 372*100*normal(x, -194.9523; 140.3033) W_B1_SingleE_Correct = 9441*100*normal(x, -151.3344; 140.2636)</p>  <p>No of obs</p>	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50.sta 45°10000c</p> <p>W_FB1_Correct = 9812*100*normal(x, -210.6865; 163.0037) W_FB1_SingleE_Real = 372*100*normal(x, -331.3273; 199.8427) W_FB1_SingleE_Correct = 9441*100*normal(x, -219.5318; 168.8265)</p>  <p>No of obs</p>

5	R & S & L & ASP & AOP	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50 sta 45v*10000c</p> <p>W_F2_Correct = 9812*200*normal(x, -259.4073, 193.7587) W_F2_SingleE_Real = 372*200*normal(x, -442.7676, 270.4764) W_F2_SingleE_Correct = 9441*200*normal(x, -270.455, 202.0691)</p> 	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50 sta 45v*10000c</p> <p>W_B2_Correct = 9812*200*normal(x, -280.525, 213.6317) W_B2_SingleE_Real = 372*200*normal(x, -336.555, 225.5858) W_B2_SingleE_Correct = 9441*200*normal(x, -295.5876, 223.4274)</p> 	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50 sta 45v*10000c</p> <p>W_FB2_Correct = 9812*200*normal(x, -468.0853, 274.9172) W_FB2_SingleE_Real = 372*200*normal(x, -604.754, 307.3031) W_FB2_SingleE_Correct = 9441*200*normal(x, -493.8512, 286.7592)</p> 
7	R & S & L &	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50 sta 45v*10000c</p> <p>W_F3_Correct = 9812*200*normal(x, -376.2303, 246.8898) W_F3_SingleE_Real = 372*200*normal(x, -553.4386, 307.9681) W_F3_SingleE_Correct = 9441*200*normal(x, -396.3542, 259.8327)</p> 	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50 sta 45v*10000c</p> <p>W_B3_Correct = 9812*200*normal(x, -412.7279, 266.1246) W_B3_SingleE_Real = 372*200*normal(x, -465.0812, 275.2804) W_B3_SingleE_Correct = 9441*200*normal(x, -436.1235, 277.8271)</p> 	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50 sta 45v*10000c</p> <p>W_FB3_Correct = 9812*500*normal(x, -717.1111, 358.9822) W_FB3_SingleE_Real = 372*500*normal(x, -843.9511, 370.4262) W_FB3_SingleE_Correct = 9441*500*normal(x, -760.2862, 375.8692)</p> 

	ASP & AOP			
9	R & S & L & ASP & AOP	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50.sta 45v*10000c</p> <p>W_F4_Correct = 9812*200*normal(x, -488.9993, 294.8676) W_F4_SingleE_Real = 372*200*normal(x, -660.4409, 342.3213) W_F4_SingleE_Correct = 9441*200*normal(x, -517.6207, 311.0049)</p> 	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50.sta 45v*10000c</p> <p>W_B4_Correct = 9812*200*normal(x, -540.2861, 306.8539) W_B4_SingleE_Real = 372*200*normal(x, -603.348, 311.1725) W_B4_SingleE_Correct = 9441*200*normal(x, -571.036, 320.2474)</p> 	<p>Histogram of multiple variables R10_S20_L30_ASP40_AOP50.sta 45v*10000c</p> <p>W_FB4_Correct = 9812*500*normal(x, -957.4383, 432.0344) W_FB4_SingleE_Real = 372*500*normal(x, -1089.2202, 430.8271) W_FB4_SingleE_Correct = 9441*500*normal(x, -1016.6653, 452.9836)</p> 



Appendix D. The Overall System GUI Description

To run the system you need the following:

1. Python 2.7 release; it can be downloaded from the following link <https://www.python.org/download/releases/2.7/>
Make sure to install 64 bit software on 64bit platform if you will use the system for tagging large data.
2. Run the system directly by clicking the system main file 'src/MainGUI.py'
3. The system parameters can be adjusted from the GUI options or from the system settings file 'src/Resources/setting.txt' as described below.

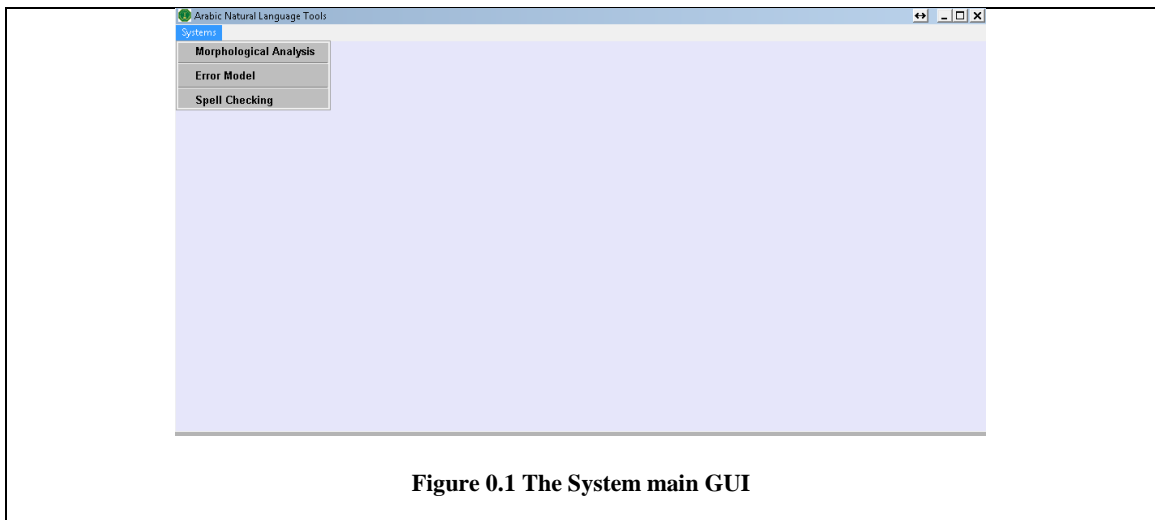


Figure 0.1 The System main GUI

The system main GUI is shown in Figure 0.1. This GUI provide access to the three developed NLP subsystems: 'the morphological analysis and disambiguation', 'the error model', and 'the general spell checking detection and correction'.

The Morphological analyzer and Disambiguater

Figure 0.2 shows the morphological analyzer and disambiguation system's main GUI. The system has three running modes: testing, evaluation and validation modes. In the testing mode, the system is used for tagging Arabic texts with a custom set of morphological features. The morphological features include the root, the stem, the lemma, the pattern and the affixes. The raw input text can be directly inserted to the GUI text area or loaded from a text file. The types of the output generated features and the output directory are determined using the setting window in Figure 0.3.

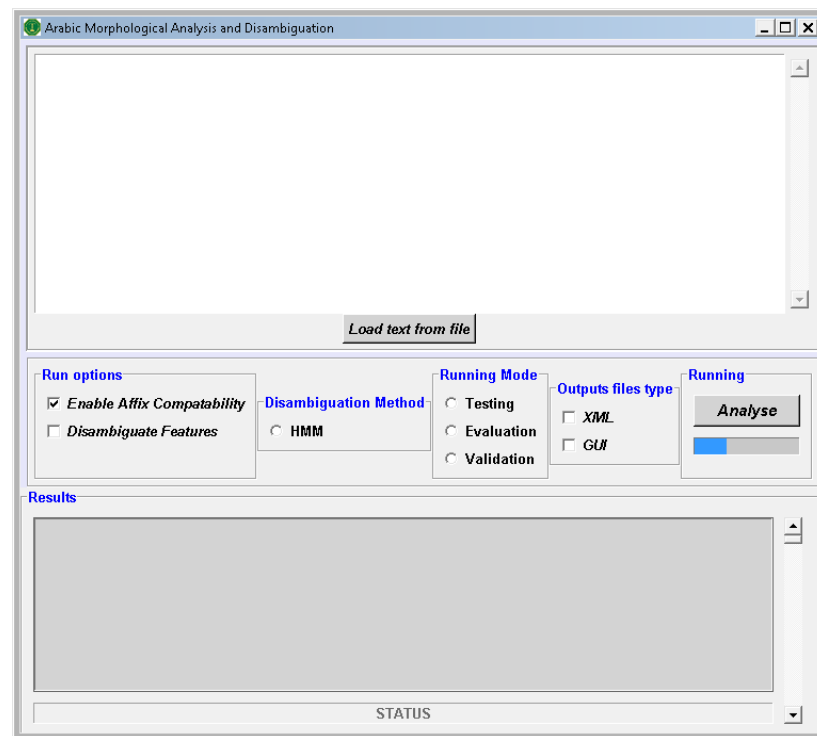


Figure 0.2 The morphological analyzer main GUI

In the evaluation mode, a morphologically tagged text file(s) should be used to evaluate the morphological analysis and disambiguation process. An analysis and summary reports

will be generated in this mode. The types of the output generated features and the output directory are determined using the setting window in Figure 0.4.

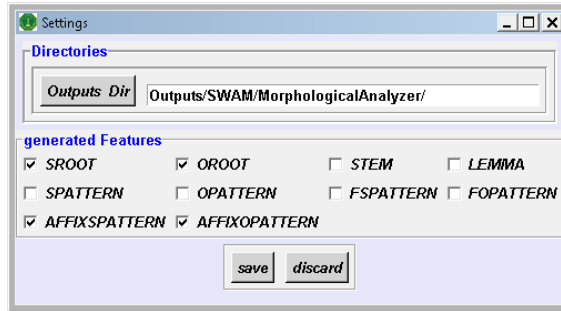


Figure 0.3 the MA output features and output directory

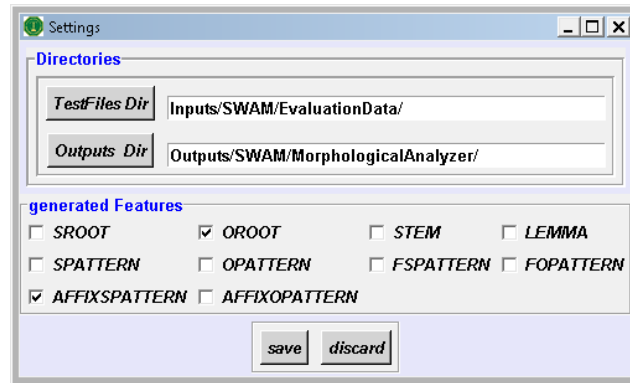


Figure 0.4 the MA output features

In this mode, the evaluation files should be formatted so that, each line should have a single word-features separated by the | character. The features should be ordered as Root(#)Stem(#)Lemma(#)AffixSPattern(#)AffixOPattern(#)FSPattern(#)FOPattern. A special word ###|### should be used for sentences boundaries. For improvement purpose, the system are implemented to generate the list of all non-recognized words with their context, this lists can be used to enrich the system lexicon data files to increase its coverage.

In the validation mode, the 10 fold cross validation algorithm, based on training corpus, is used to provide the best parameters values for the system (lambda). A report will be generated in this mode to show the best parameter values to be used.

The morphological analyzer use HMM model with Viterbi algorithm for features disambiguation. The model can run based on different feature(s)/states. The types of the features with their parameters are adjusted from the hmm window as shown in Figure 0.5.

Lambda ranges are to be used for evaluation mode only.

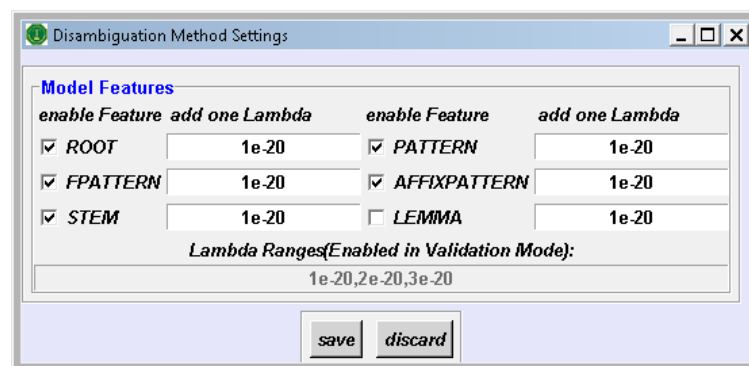


Figure 0.5 the HMM model setup

According to the morphological analyzer settings, the analysis results will be generated in the specified output directory. The generated files include: the result summary file, the input words with all their possible analysis (morphological_analysis_details.xml), and the input words with their best analysis only (morphological_analysis_with_disambiguation_details.xml). The analysis files in xml format as shown in Figure 0.6.

```

<?xml version="1.0" encoding="utf-8"?>
<Text>
<Sentence text="نعمد بالتحليل اللغوي تفكيك الظاهرة اللغوية إلى عناصرها الأولية التي تتألف منها" ... ">
  <Word hasMA="True" numberOfPossibleAnalyses="1" value="نعمد">
    <analysis type="1">
      <feature originalRoot="نعمد"/>
      <feature suffix=""/>
      <feature surfaceRoot="نعمد"/>
      <feature originalPattern="فعل"/>
      <feature SurfacePattern="فعل"/>
      <feature lemma="نعمد"/>
      <feature prefix=""/>
      <feature stem="نعمد"/>
    </analysis>
  </Word>
  <Word hasMA="True" numberOfPossibleAnalyses="1" value="بالتحليل">
    <analysis type="1">
      <feature originalRoot="حلل"/>
      <feature suffix=""/>
      <feature surfaceRoot="حلل"/>
      <feature originalPattern="تفعليل"/>
      <feature SurfacePattern="تفعليل"/>
      <feature lemma="تفعليل"/>
      <feature prefix="بال"/>
      <feature stem="تفعليل"/>
    </analysis>
  </Word>
  <Word hasMA="True" numberOfPossibleAnalyses="7" value="اللغوي">
    <analysis type="1">
      <feature originalRoot="لغو"/>
      <feature suffix=""/>
      <feature surfaceRoot="لغوي"/>
      <feature originalPattern="فحول"/>
      <feature SurfacePattern="فحول"/>
    </analysis>
  </Word>
</Sentence>
</Text>

```

Figure 0.6 the morphological analysis result file

The Error Model

The GUI in Figure 0.7 shows the error model main GUI. The error model generates candidates' corrections for any word based on an already learnt error patterns.

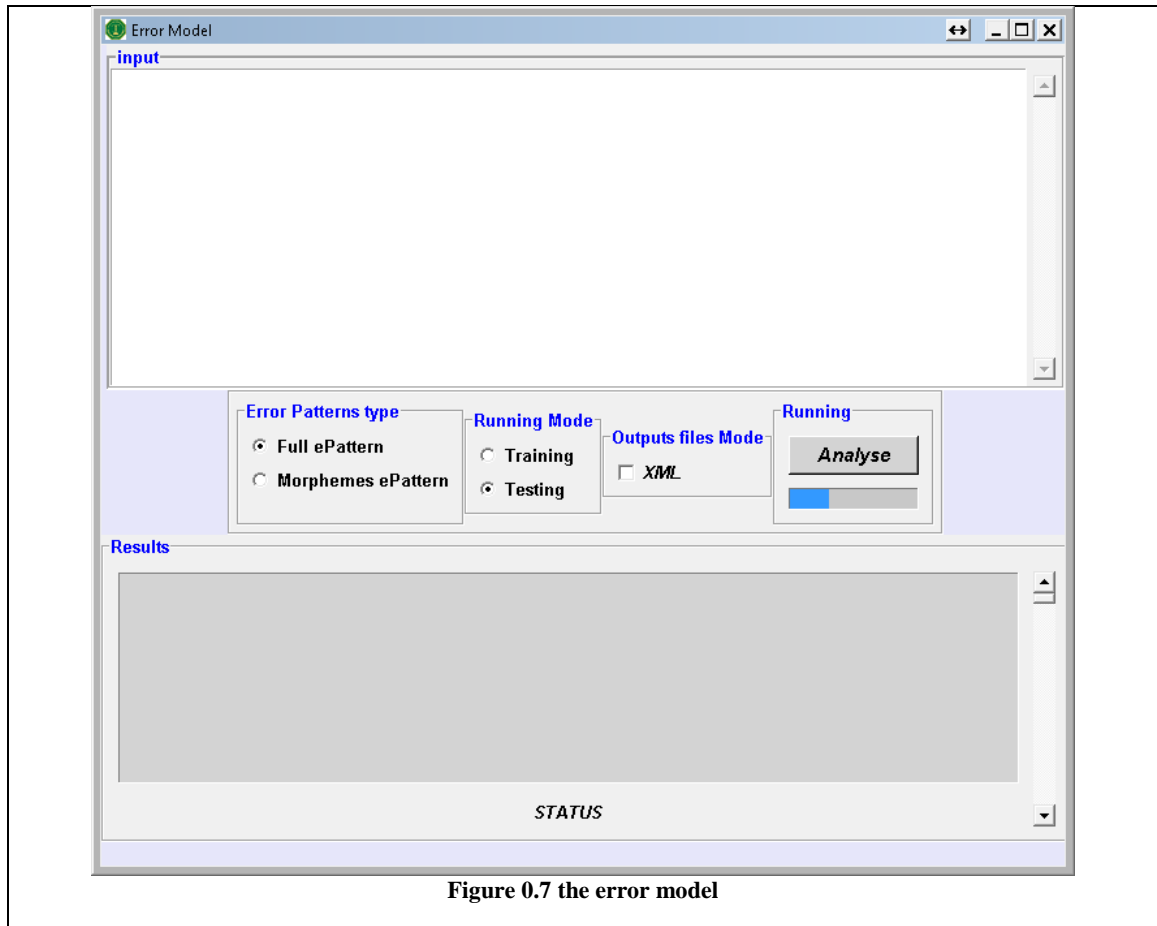


Figure 0.7 the error model

The error model has two running modes: training and testing modes. In the training mode, all the needed correction information will be learned from an already tagged error corpora. The statistics of the training corpora will be generated in this mode, the statistics can provides detailed analysis of the corpus errors as explained in the error model report (see the error model theory report). The training file's input and output directories are determined through the GUI in Figure 0.8.

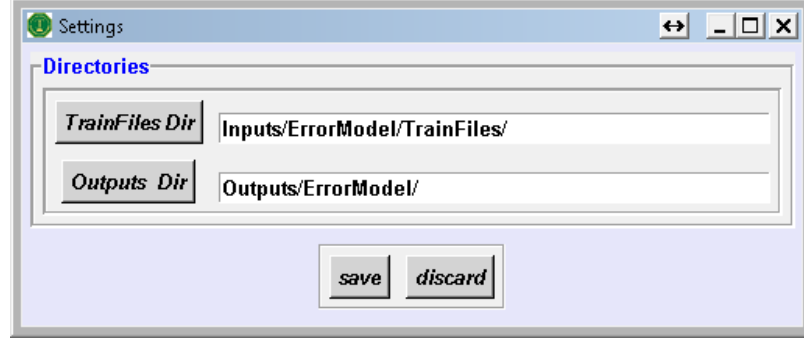


Figure 0.8 the error model train files and output directories

The input train files should be formatted²⁸ so that, each line should have the error word, the affix correct word and the correct word's affix pattern separated by the | character, a special word ### should be used for paragraphs boundaries.

The learnt error patterns will be saved in XML files. The xml files stores error patterns based on their length, each file correspond to the error patterns with specific length. Each file store the error patterns along with all their correction information. A simple snapshot of the result file is shown in Figure 0.9.

```
<Error_Pattern value="مفأ$أ">
|
| <Correction correct_Pattern="مفأ$أ" error_Code_Chars="ف$أ" error_code="$---$-$" error_pattern_frequency="34"/>
| </Error_Pattern>
| <Error_Pattern value="أفأ$أ">
| | <Correction correct_Pattern="أفأ$أ" error_Code_Chars="أ$أ" error_code="$s---$-$" error_pattern_frequency="1784"/>
| | </Error_Pattern>
| <Error_Pattern value="مفأ$أ">
| | <Correction correct_Pattern="مفأ$أ" error_Code_Chars="ف$أ" error_code="$---$s$" error_pattern_frequency="3"/>
| | </Error_Pattern>
| <Error_Pattern value="فأأ$أ">
| | <Correction correct_Pattern="فأأ$أ" error_Code_Chars="ف$أ" error_code="$--s-$-$" error_pattern_frequency="75"/>
| | </Error_Pattern>
| <Error_Pattern value="أفأ$أ">
| | <Correction correct_Pattern="أفأ$أ" error_Code_Chars="ف$أ" error_code="$---$-$" error_pattern_frequency="194"/>
| | </Error_Pattern>
| <Error_Pattern value="أفأ$أ">
| | <Correction correct_Pattern="أفأ$أ" error_Code_Chars="أ$أ" error_code="$s---$-$" error_pattern_frequency="49"/>
| | <Correction correct_Pattern="أفأ$أ" error_Code_Chars="ف$أ" error_code="$d$---$-$" error_pattern_frequency="60"/>
| | </Error_Pattern>
| <Error_Pattern value="أفأ$أ">
| | <Correction correct_Pattern="أفأ$أ" error_Code_Chars="ف$أ" error_code="$---$d$-$" error_pattern_frequency="186"/>
| | <Correction correct_Pattern="أفأ$أ" error_Code_Chars="ف$أ" error_code="$---$d$-$" error_pattern_frequency="1"/>
| | </Error_Pattern>
```

Figure 0.9 the error patterns result file snapshot

²⁸ To generate the error model input format for QALB corpus, you should separately run the file `views/analyzeQALBData.py`.

In the testing mode, the list of correction candidates are displayed for any input word based on the learnt error patterns of the training mode. The function in the testing mode can be used by any spelling correction for generating correction candidates.

The error model works at word and morphemes level. In the word level, the error pattern of the whole word is generated and considered in the candidate's generation process. In the morphemes level, all the possible morphemes error patterns are generated and considered in the candidates' generation process.

The General Spell Checking Detection and Correction

Figure 0.10 shows the main GUI of the general spell checker. The main components for the system are: errors' detection and error correction. Error detection component is responsible for detecting suspected errors in the input text. Error correction component is responsible for generating a list of all probable corrections for each of the detected errors and then selects the most appropriate correction from the list of the generated candidates.

The system handles non-word and real-word spelling errors using different techniques. It also has three running modes: testing, evaluation and validation modes. In the testing mode, the system is used for the detection and correction of Arabic text errors. The input text can be directly inserted to the GUI text area or loaded from a text file. In the evaluation mode an error annotated text files are used to evaluate the spell checking detection and correction techniques. The input files should be formatted so that, each line has a single word-annotation separated by the | character (word|correction|errortype), a special word ### should be used for sentences boundaries. The system, in the evaluation mode, is able to deal with two different annotated formats: KFUPM or KACST format. In the validation

mode, a 10-fold cross validation algorithm is used to generate the best parameters values for the system (lambda, window sizes and thresholds).

The input and output setting are determined using the window in Figure 0.11 .

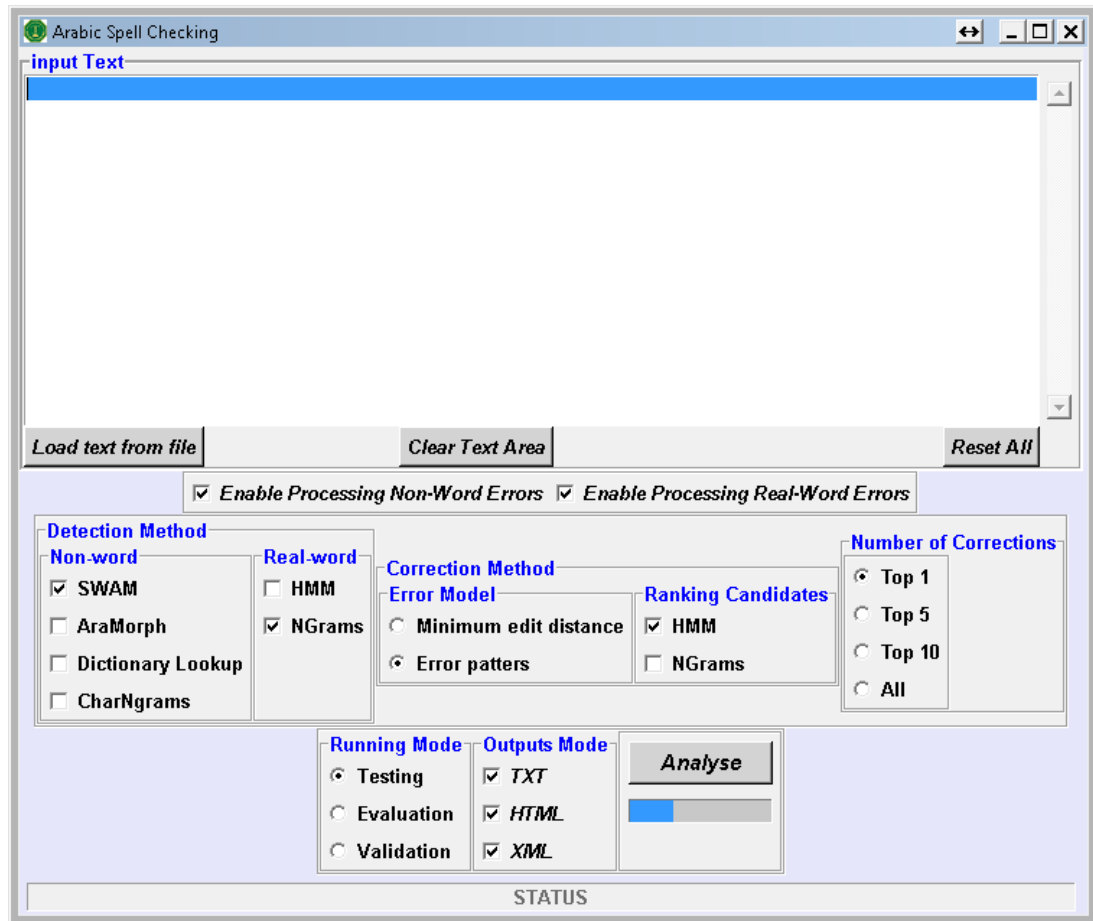
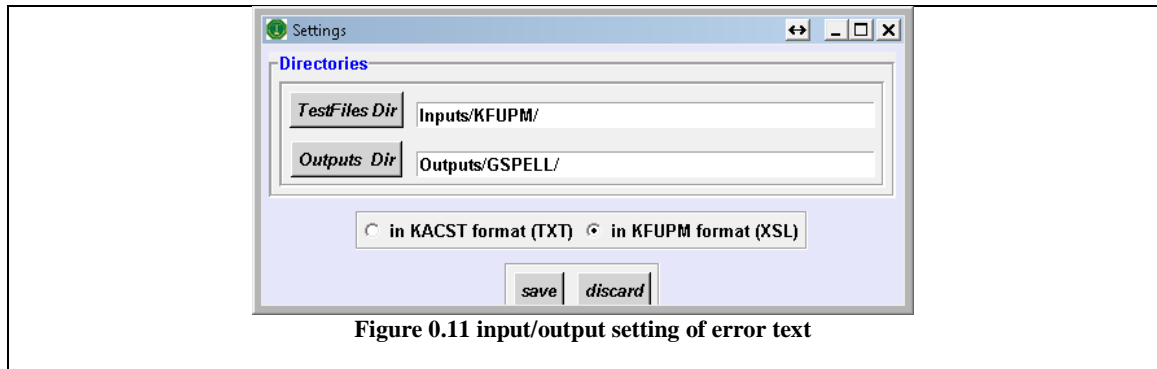
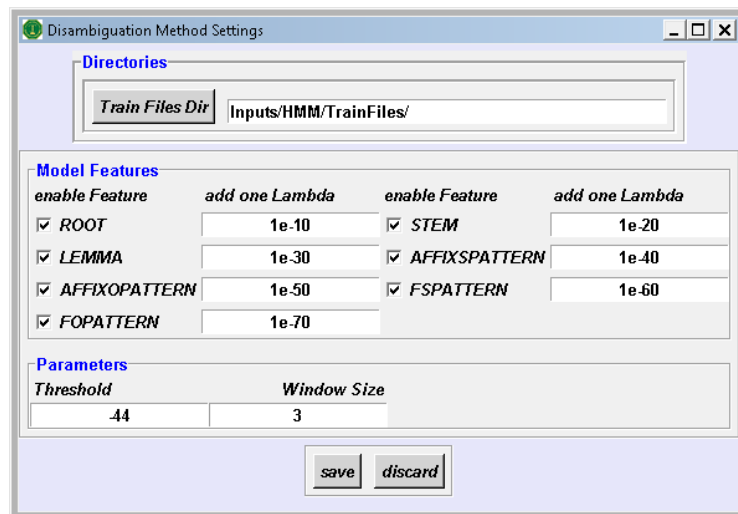


Figure 0.10 The GSpell main GUI



In the validation mode, a validation algorithm is used to generate the best parameter's values for the system (lambda, window sizes and thresholds). A report will be generated in this mode to show the best parameter's values to be used in the testing and evaluation mode. In the case of HMM method, the model settings can be set using the window in Figure 0.12.



The provided threshold value should be the log of the threshold probability.

The system can handle different types of errors: non-word and real-word spelling errors.

Different methods can be used to handle each type of spelling errors.

Non-word Errors Detection

To detect the erroneous words in the input text, the error detection module implements the following techniques: morphological analyzers (SWAM and Aramorph), dictionary look-up, and language model (character N-grams). The system is implemented to allow using these techniques individually or in combinations.

SWAM

A morphological analyzer that uses a Sliding Window Asynchronous Matching (SWAM) approach has been implemented. SWAM is a lexicon driven approach that uses morphological derivational forms (window patterns) to extract the probable morphological feature tuples for any given input word. The generated morphological features include the root, the stem, the lemma, the morphological pattern and the affixes. Disambiguation and ambiguity are resolved a markovian based Viterbi algorithm. Any word in the input text that has no morphological analysis is considered as Non-word error.

Buckwalter Arabic Morphological Analyzer

Buckwalter Arabic Morphological Analyzer is rule-based morphological analyzer that depends on a set of lexicon lists. It depends on three main elements: the data, the compatibility tables and the morphological analysis algorithm. The data is composed of three lexicon files: prefixes, suffixes and stems. It contains three morphological compatibility tables to validate prefix-stem, stem-suffix, and prefix-suffix combinations. Any word in the input text that has no morphological analysis is considered as Non-word error.

Dictionary look-up

A dictionary look-up technique is used for validating input words. Input word is looked up in the dictionary. If that word is in the dictionary, then it is assumed as a correct word. Otherwise, it is considered as an erroneous word.

A set of dictionaries is generated from a set of collected corpora that covers different subjects. The dictionaries are generated based on a different the word frequencies which are then validated using SWAM. Table 0-6 shows the corpora used for generating the dictionary. These corpora are not used in the manually annotated project corpus.

Table 0-6 Collected and used corpora

Corpus	Size	Source
Health	28.8 MB	Al-Riyadh
Sport	129.8 MB	Al-Riyadh
Economics	7.13 MB	Al-Riyadh
Collection1	127 MB	(Mahdi 2012)
Collection2	4.2 MB	(Yemeni Newspapers)
General	296.93 MB	

Table 0-7 show the statistics of the generated dictionaries after they are morphologically checked by SWAM. The currently used dictionary in the system is Dictionary 3.

Table 0-7 Statistics of dictionaries

Dictionary	Minimum # of occurrences	Dictionary size in words
Dictionary 1	1	376116
Dictionary 2	2	234728
Dictionary 3	5	125975
Dictionary 4	10	95273
Dictionary 5	20	64751
Dictionary 6	50	36498
Dictionary 7	100	23275

Character N-Grams

Character N-grams is another technique that is provided by the system to detect non-word errors. Character n-grams are a subsequence of n characters of a word. This technique

works as follows: for each character n-gram in an input word, a pre-compiled table of n-gram statistics is searched to determine its existence. Building language model of character n-grams requires a large, verified corpus of texts. In this work, an n-gram (n=2-4) model is generated for this purpose.

The statistics of the generated character bi-grams, tri-grams and quad-grams by (Mahdi, 2012) are shown in Table 0-8.

Table 0-8 Character N-grams

n-gram	Token	Type (Distinct)
Bi-grams	2,634,535	1,110
Tri-grams	2,276,731	18,633
Quad-gram	1,918,927	129,053

Real word spell checking Ngrams

The problem of real-word errors in Arabic text is addressed using an unsupervised technique in which n-gram language models are used to detect and correct real-word errors. For the detecting real-word error, the system finds suspicious words in a given text by checking the availability of tri-grams in a sentence. If not found the module further checks for the availability of the two bi-grams in the three words trigram. If not found, the word in the middle is considered suspicious. In order to check whether the suspicious word is an error, all its spelling variations are retrieved and put in the sentence. If the probability of the word in the sentence is high, the suspicious word is an error. Table 3 shows the statistics of the corpus used to generate the language models.

Table 0-9 LMs statistics of the corpus

Number of words	Uni-grams	Bi-grams	Tri-grams
26,879,902	507,722	9,139,710	2,345,283

HMM

The probability of the inspected word to happen in some context is highly related to the probability of its morphological features in the same context. This probability is computed for each word based on its morphological features using a markovian based Viterbi algorithm. Any word with morphological features probability “less than a threshold” is flagged as suspected real-word error.

The supervised technique

The problem of real-word errors in Arabic text is implemented using context words and n-gram language models using collection of confusion sets.

The probabilities of the context words of the confusion sets are estimated using a window-based technique. N-gram language models are used to detect real-word errors and to choose the best correction for the errors once found. The two prototypes were implemented in Python in addition to the baseline module to compare with.

Context words method

This method uses the context words surrounding the target words from predefined confusion sets. We identify words that are semantically unrelated to their context. Then, we find which of the word variations, from the confusion set, is more related to that context and could be the best replacement for the erroneous word. Maximum likelihood estimate is used to estimate the probabilities of the context words surrounding the target word. The probabilities for every word in the confusion set are calculated. The word in the confusion

set with the highest probability is chosen. It should be mentioned this part is implanted and evaluated separately.

N-Gram Method:

N-gram language models were used to disambiguate words of the confusion sets using the sequence of the words in the context in which they appear. For each target word in the confusion set, the words surrounding it are used to predict the proper word in that sentence. For each word in the confusion set, a new sentence will be generated by placing the confusion set word in place of the target word. The probabilities of all the sentences with respect to the confusion set words are calculated. In case that the tri-gram is not found, bi-grams back off is used and uni-gram back off is used when a bi-gram is not found. The sentence that gives the highest probability is considered as the correct one indicating that the confusion set member in that sentence is a better choice and hence more likely to be the correct word.

Error Correction Techniques

To correct the errors detected by the aforementioned error detection techniques for an input text. The correction prototype is implemented to include: generating a list of candidate words and ranking the candidate words.

Candidates generation techniques

Two different techniques are used for candidates' generation namely minimum edit distance and error pattern.

Minimum Edit Distance

This technique is also called, Damerau-Levenshtein distance. Damerau-Levenshtein algorithm computes the minimum number of editing operations (adding, replacing or

deleting) required to change one word into another. The system is implemented to add, replace or delete n-characters ($n=1$ or 2) to the misspelled words. The correctness of the generated words is checked; i.e., if a generated word is in the dictionary it will be taken as a candidate word.

Error Pattern

A data driven system that exploits morphological error patterns at morphemes or word levels is implemented. Its main components are the error-correct patterns generator (ECPG), the error-correct patterns database (ECPD) and the correction candidates' generator (CCG). The ECPG is a module that generates morphological error patterns and their correction information to be used for the correction process. The information generated by the ECPG is used to build the ECPD so that it will be later used by the CCG to generate the correction candidates.

Ranking Candidates Techniques

The system provides different methods to rank the candidates based on their probabilities which are HMM or/ and N-grams.

HMM

After the correction candidates with their possible morphological features are generated for each suspected error word. The word with the highest morphological features probability in the context is selected by the correction algorithm as the best correction. It should be mentioned that this technique works only if SWAM is selected among a detection techniques.

N-grams

After the correction candidates are generated, the correction that gives the highest probability in the context is compared with the original suspect word. If the correction probability in the context is higher than the original probability (with respect to a threshold value), then the correction is more likely to be the intended word and the original word is replaced with that variation.

HMM and N-grams

Both techniques can be used for ranking candidates based on a ranking mechanism. This mechanism works as follow. After generating the candidates of each technique separately; the candidates are given a rank in descendent order. The new combined ranked candidates are now ordered based on the weighted total sum of ranks in each technique. The weight of each technique are evaluated from validation set. In our case, the weights for HMM techniques is 0.2230 and the weight of NGrams techniques is 0.3128.

It needs to be mentioned that, the system is implemented to allow the user to determine the number of candidates that the system should present. The effectiveness of the system can be evaluated using different candidates.

The Representation of the Results

The system provides three formats for the outputs:

1. Text file: when this option is selected the system generates the text file with format of “.txt” which contains the corrected text with selecting the top one candidate.
HTML: when this option is selected the system creates HTML file with format of “.html” which contains the corrected text with presenting different candidates. The number of candidates is determined by the user.

2. XML: when this option is selected the system creates XML file with format of “.xml” which contains the corrected text with presenting different candidates. The number of candidates is determined also by the user.

Additionally, a summary.txt file is generated to report the results. It is essential for the evaluation mode and it contains several evaluation metrics including number of errors, Detection/Correction true Positives (DTP, CTP), Detection/Correction False Positives (DFP, CFP), Detection/Correction False Negative (DFN, CFN), Recall, precision and F1-measure which are the common natural language processing measures. The correction measures are computed for different number of corrections (the n-top candidates) determined by the user. Moreover, the list of error words are generated in a separate file named “detectedwords.txt”.

The options in the morphological analyzer GUI can also be adjusted using the system setting file 'src/Resources/settings.txt', the setting file has a set of variable that control the system functions. The following are a short description of the system variables.

SWAM_DB_FILES_DIR=./Inputs/SWAM/DB/NewTextFiles/

The directory of the SWAM lexicon files, the lexicon files includes: Prefixes, Suffixes, SurfaceRoots, OriginalRoots, SurfacePatterns, OriginalPatterns, SurfaceInvariables, OriginalInvariables, SurfaceOriginalRoots, SurfaceOriginalPatterns, and SurfaceOriginalInvariables.

SWAM_AFFIXES_TRAIN_FILES_DIR=Outputs/SWAM/AffixCompatability/PatternsAffixCompatability.xml

The patterns' affixes compatibility file path, it is an xml file that store all the templatic and non-templatic patterns with their compatible affixes.

SWAM_TestFiles_Dir=Inputs/SWAM/EvaluationData/

The directory of the evaluation file(s), the evaluation file should be formatted so that, each line has a single word-features separated by the | character. The features should be ordered as Root(#)Stem(#)Lemma(#)AffixSPattern(#)AffixOPattern. A special word ###|### should be used for sentences boundaries.

SWAM_OUTPUT_Dir=Outputs/SWAM/MorphologicalAnalyzer/

The system output directory.

ANALYZED_FEATURES_TYPES=Templatic,NonTemplatic

This specify the types of features that the system will provide the results for (It is used only in evaluation mode). In this mode, the evaluation file should be formatted so that, each line has a single word-featureType-features separated by the | character. The features should be ordered as Root(#)Stem(#)Lemma(#)AffixSPattern(#)AffixOPattern. A special word ###|### should be used for sentences boundaries.

SWAM_OUTPUT_FEATURES=ROOT,SPATTERN,OPATTERN,FSPATTERN,FOPATTERN,STEM,LEMMA,AFFIXSPATTERN,AFFIXOPATTERN,ROOT&AFFIXSPATTERN,ROOT&AFFIXOPATTERN,ROOT&FSPATTERN,ROOT&FOPATTERN

The set of features and features combinations that the system will provide for system improvement.

HMM_TRAIN_FILES_Dir=Outputs/NEMLAR/Train/

The directory of the model train file(s), the train file should be formatted so that, each line has a single word-features separated by the | character. The features should be ordered as Root(#)Stem(#)Lemma(#)AffixSPattern(#)AffixOPattern . A special word ###/### should be used for sentences boundaries.

**HMM_USED_FEATURES=ROOT,STEM,LEMMA,AFFIXSPATTERN,AFFIXOP
ATTERN,FSPATTERN,FOPATTERN**

The features that the model will use for disambiguation process.

HMM_USED_ADD_ONE_LAMBDA=1e-20,1e-20,1e-20,1e-20,1e-20

The corresponding lambda values for the HMM_USED_FEATURES.

HMM_LAMBDA_RANGES=1e-20,2e-20,3e-20

The list of lambda values to be used in the validation modes, the reported best lambda parameter should be used in the evaluation and testing phases.

ERROR_MODEL_RUNING_MODE=TESTING

The error model running mode. The system can run in either TRAINING or TESTING modes.

ERROR_MODEL_EPATTERNS_LEVEL=FULL_EPATTERN_LEVEL

The level of the error patterns. Two levels are supported by the system:

FULL_EPATTERN_LEVEL and MORPHEMES_EPATTERNS_LEVEL.

ERROR_MODEL_ERROR_ANNOTATED_TRAINFiles_Dir

=Inputs/ERROR_MODEL/QALB/Dev/

The directory of the train file(s), the train file should be formatted so that, each line has a single word-annotation separated by the | character, a special word ### should be used for sentences boundaries. This files are generated using a separate module.

ERROR_MODEL_MORPHOLOGICAL_ERROR_PATTERNS_Dir

=Inputs/ErrorModel/ErrorPatterns/

The directory of the already built error patterns in xml format, the system will load all the files and will be ready for testing mode.

ERROR_MODEL_OUTPUT_DIR =Outputs/ErrorModel/

The output directory of the learnt error patterns and statistics

GSPELL_TestFiles_Dir=Inputs/GSpell/evaluation/

The directory of the test file(s), the test file should be formatted so that, each line in the file has a single word-annotation separated by the | character, (input word |correct word error type), a special word ### was used for paragraph boundaries.

GSPELL_INPUT_DATA_FORMAT=KACST_TXT

The format of the input files. Two formats are supported by the system:

KACST_TXT or KFUPM_XSL

GSPELL_OUTPUT_Dir=Outputs/GSPELL/

The output directory of the result files; three files are generated: summary, result.xml, result.html

GSPELL_RUN_MODE=TESTING

The system running mode. The system can run in TESTING ,EVALUATION and VALIDATION. The default is the TESTING mode.

GSPELL_ENABLE_PROCESSING_REALWORD_ERRORS=1

1 to enable processing real-word errors, 0 to disable

GSPELL_ENABLE_PROCESSING_NONWORD_ERRORS=1

1 to enable processing non-word errors, 0 to disable

GSPELL_NONWORD_ERRORS_DETECTION_METHOD=SWAM

Specify the non-word detection method: SWAM, Aramorph, Dictionary or CharNgrams

GSPELL_REALWORD_ERRORS_DETECTION_METHOD=HMM

Specify the non-word detection method: HMM, NGRAMS

GSPELL_ERRORS_CORRECTION_METHOD=HMM

Specify the error correction method: HMM or NGRAMS.

GSPELL_ERROR_MODEL_TYPE=ERROR_PATTERNS

Specify the type of the used error model: ERROR_PATTERNS or minimum edit distance

GSPELL_NUMBER_OF_RESULT_CORRECTIONS=TOP_1

Specify the number of the corrections to consider for each error word: Top_1, Top_5, Top_10 or Top_x. Top_X means all the corrections list.

GSPELL_HMM_DETECTION_THRESHOLD=-44

Specify the HMM detection threshold, this threshold are used for the detection of real word errors.

GSPELL_HMM_DETECTION_WINDOW_SIZE=3

Specify the HMM detection window size, this window specify how many context words to consider in the left and right of the error word.

GSPELL_HMM_CORRECTION_WINDOW_SIZE=3

In the ranking of candidates' correction, this value specify how many context words to consider in the left and right for ranking each candidate.

Vitae

Name : Tamim Salah Abdallah Alnethary

Nationality : Yemeni

Date of Birth : 1/1/1984

Email : alnetharytamim@yahoo.com

Address : Taiz - Yemen

Academic Background : BS degree in information technology, Taiz University,
Yemen ,2008

PUBLICATIONS:

- Morphological Analysis and Disambiguation Using HMMs, paper accepted for SSC7 student conference 2016.